On the Theory and Applications of Hierarchical Learning Automata and Object Migration Automata

Rebekka Olsson Omslandseter

UiA University of Agder

Doctoral Dissertations at the University of Agder 444

Rebekka Olsson Omslandseter

On the Theory and Applications of Hierarchical Learning Automata and Object Migration Automata

Dissertation for the degree philosophiae doctor (ph.d.)
at the Faculty of Engineering and Science, Specialization in Artificial Intelligence

University of Agder
Faculty of Engineering and Science
2023

Doctoral Dissertations at the University of Agder 444

ISSN: 1504-9272
ISBN: 978-82-8427-161-3

Printed by Make!Graphics
Kristiansand

Preface

This dissertation is a result of the research work carried out at the Department of Information and Communication Technology (ICT), University of Agder (UiA) in Grimstad, Norway, from January 2019 to September 2023. I started my Ph. D. as an integrated Ph. D. student, working 50% with the Master's Education, and 50% with the Ph. D. until June 2020. My main supervisor has been Dr. Lei Jiao, University of Agder, and my co-supervisor has been Chancellor's Professor Dr. B. John Oommen, Carleton University, in Ottawa, Canada. B. John Oommen is also an Adjunct Professor with the University of Agder.

Production note: This dissertation, as well as my papers produced during my Ph. D. study, have been written using LaTeX. The mathematical calculations and simulation results are obtained using MATLAB, and the Pycharm/Spyder IDE for Python scripts. To make the visualizations in this dissertation, we mainly utilized the visualization tool Lucid and the online math editor Mathcha.

Acknowledgments

I want to express my gratitude to my two supervisors, Dr. Lei Jiao and Dr. B. John Oommen, for their guidance, support, and their mentorship throughout my Ph.D. journey. Our collaboration has been highly valuable, and made it possible for the Ph.D. work to maintain the expected quality during the whole process. My sincere gratitude goes out to them for providing me with direction and helping me to improve my research and scientific contributions. Their assistance and guidance have not only been academic and scientific. Indeed, they have taught me the invaluable art of doing research in new domains, exploring the state of the art, determining new solutions, implementing and testing them, and then, taking these results through the difficult process of being refereed and published in acclaimed venues. I am grateful to them for my whole life.

I would also like to express my heartfelt appreciation to my absolutely wonderful family and friends, who have been a steady source of encouragement and inspiration to me. I am incredibly grateful for their understanding, love, and unwavering support during this Ph.D. journey. I am immensely grateful for your strong belief in me and the dedicated support you have provided me throughout the challenges I faced during this process. You are the best!

Additionally, I want to express my sincere gratitude to the incredible individuals at UiA and in CAIR who have played a very important role in this journey. Their eagerness to share their knowledge, guidance, and encouragement have been invaluable.

Finally, I would like to express my genuine gratitude to the companies Bouvet and Glitre Nett for providing me with the opportunity to work part-time on extremely interesting projects during my Ph.D. studies. The practical insights and experiences gained through these engagements have enhanced my technical skills, expanded my interests, and offered invaluable insights into real-world challenges and applications.

Last, but not least, I am humbled and truly grateful for all of the support and contributions of all those mentioned above, as well as all those who I have not explicitly acknowledged here but that have played a part in my academic journey in any way. Your encouragement has been invaluable, and I sincerely appreciate every one of you.

Rebekka Olsson Omslandseter
Grimstad
August 8, 2023

Abstract

The paradigm of Artificial Intelligence (AI) and the sub-group of Machine Learning (ML) have attracted exponential interest in our society in recent years. The domain of ML contains numerous methods, and it is desirable (and in one sense, mandatory) that these methods are applicable and valuable to real-life challenges. Learning Automata (LA) is an intriguing and classical direction within ML. In LA, non-human agents can find optimal solutions to various problems through the concept of *learning*. The LA instances *learn* through Agent-Environment interactions, where advantageous behavior is rewarded, and disadvantageous behavior is penalized. Consequently, the agent eventually, and hopefully, learns the optimal action from a set of actions. LA has served as a foundation for Reinforcement Learning (RL), and the field of LA has been studied for decades. However, many improvements can still be made to render these algorithms to be even more convenient and effective. In this dissertation, we record our research contributions to the design and applications within the field of LA.

Our research includes novel improvements to the domain of hierarchical LA, major advancements to the family of Object Migration Automata (OMA) algorithms, and a novel application of LA, where it was utilized to solve challenges in a mobile radio communication system. More specifically, we introduced the novel Hierarchical Discrete Pursuit Automaton (HDPA), which significantly improved the state of the art in terms of effectiveness for problems with high accuracy requirements, and we mathematically proved its ϵ-optimality. In addition, we proposed the Action Distribution Enhanced (ADE) approach to hierarchical LA schemes which significantly reduced the number of iterations required before the machines reached convergence.

The existing schemes in the OMA paradigm, which are able to solve partitioning problems, could only solve problems with equally-sized partitions. Therefore, we proposed two novel methods that could handle unequally-sized partitions. In addition, we rigorously summarized the OMA domain, outlined its potential benefits to society, and listed further development cases for future researchers in the field.

With regard to applications, we proposed an OMA-based approach to the grouping and power allocation in Non-orthogonal Multiple Access (NOMA) systems, demonstrating the applicability of the OMA and its advantage in solving fairly complicated stochastic problems. The details of these contributions and their published scientific impacts will be summarized in this dissertation, before we present some of the research contributions in their entirety.

Sammendrag

Feltet kunstig intelligens og undergruppa maskinlæring har oppnådd en eksponentiell interesse i samfunnet vårt over de seneste årene. Innenfor feltet maskinlæring finner vi en mengde ulike metoder, og det blir stadig viktigere at disse metodene er brukbare og verdifulle i de utfordringene vi mennesker møter og i oppgaver vi ønsker løst av datamaskiner. Feltet som omhandler læringsmaskiner, er et fascinerende og klassisk felt innen maskinlæring. En læringsmaskin er en ikke-menneskelig agent som kan finne optimale løsninger på forskjellige problemer ved hjelp av konseptet *læring*. Læringsmaskinene lærer via agent-miljø interaksjoner, der fordelaktig oppførsel belønnes og ufordelaktig oppførsel straffes. På denne måten vil læringsmaskinene etter hvert, og forhåpentligvis, klare å finne den optimale handlingen fra et sett av potensielle handlinger den kan utføre. Læringsmaskin-feltet har figurert som selve fundamentet for forsterkningslæring, og feltet har blitt studert i årtier. Likevel er det mange forbedringer som kan gjøre disse algoritmene enda mer nyttige og effektive. I denne avhandlingen oppsummerer vi våre forskningsbidrag på læringsmaskins-feltet.

Vår forskning inkluderer banebrytende forbedringer på hierarkisk læringsmaskins-feltet, store forbedringer på Object Migration Automata (OMA)-algoritmene, og en helt ny måte å løse utfordringer innen et mobilradiokommunikasjonssystem ved hjelp av læringsmaskins-metoder. Mer spesifikt så introduserte vi Hierarchical Discrete Pursuit Automaton (HDPA) og denne læringsmaskinen var betraktelig bedre enn de eksisterende sammenliknbare algoritmene for problemer der det er høye krav til nøyaktighet. Vi beviste også matematisk at HDPA-algoritmen var ϵ-optimal. I tillegg foreslo vi Action Distribution Enhanced (ADE)-tilnærmingen til hierarkiske læringsmaskiner, som betydelig reduserte antall iterasjoner som krevdes før maskinene nådde konvergens. De eksisterende algoritmene i OMA-paradigmet, som er i stand til å løse grupperingsproblemer, kunne bare løse problemer med like store grupper. Derfor foreslo vi to nye metoder som kunne håndtere gruppe-konstellasjoner med ulik størrelse. I tillegg oppsummerte vi OMA-domenet grundig, skisserte dets mulige fordeler for samfunnet og listet opp lovende nye retninger for fremtidige forskere på feltet. Videre foreslo vi en OMA-basert tilnærming til gruppering og ressursallokering i Non-ortogonal Multiple Access (NOMA) systemer, og demonstrerte på denne måten anvendeligheten til OMA og dens fordel i å løse stokastiske problemer. Detaljene i disse forskningsbidragene vil bli oppsummert i denne avhandlingen, før vi presenterer noen av forskningsbidragene i sin helhet.

Publications

The author of this dissertation is the primary contributor and is responsible for all the papers listed below. The papers from A-J, in the first set, are selected to represent the main contributions of the research and are reproduced in Part 2 of the dissertation. Paper 11 in the second set is complementary to the main research contributions. The reader should note that the level in the Norwegian system is indicated in brackets after the paper id.

Papers Included in the Dissertation[1]

Paper A (1) *Conference Paper:* R. O. Omslandseter, L. Jiao, and J. B. Oommen, "A Learning-Automata Based Solution for Non-equal Partitioning: Partitions with Common GCD Sizes," *Advances and Trends in Artificial Intelligence, From Theory to Practice, IEA/AIE 2021*, vol 12799, pp. 227–239, Springer International Publishing, July 2021.
https://doi.org/10.1007/978-3-030-79463-7_19

Paper B (1) *Conference Paper:* R. O. Omslandseter, L. Jiao, and J. B. Oommen, "Object Migration Automata for Non-Equal Partitioning Problems with Known Partition Sizes," *Artificial Intelligence Applications and Innovations, AIAI 2021*, vol 627, pp. 129–142, Springer International Publishing, June 2021.
https://doi.org/10.1007/978-3-030-79150-6_11

Paper C (1) *Journal Paper:* J. B. Oommen, R. O. Omslandseter, and L. Jiao, "Learning Automata-Based Partitioning Algorithms for Stochastic Grouping Problems with Non-Equal Partition Sizes," *Pattern Analysis and Applications*, vol 26, pp. 751–772, Springer London, May 2023.
https://doi.org/10.1007/s10044-023-01131-5

Paper D (1) *Journal Paper:* J. B. Oommen, R. O. Omslandseter, and L. Jiao, "The Object Migration Automata: Its Field, Scope, Applications, and Future Research Challenges," *Pattern Analysis and Applications*, Special Issue, pp. 1–12, Springer London, April 2023.
https://doi.org/10.1007/s10044-023-01163-x

[1]The papers are not listed chronologically, but in the same order as they are presented in the Table of Contents.

Paper E (1) *Conference Paper:* R. O. Omslandseter, L. Jiao, X. Zhang, A. Yazidi, and J. B. Oommen, "The Hierarchical Discrete Learning Automaton Suitable for Environments with Many Actions and High Accuracy Requirements," *AI 2021: Advances in Artificial Intelligence, AI 2022 (AJCAI 2021)*, vol 13151, pp. 507–518, Springer International Publishing, February 2022.
https://doi.org/10.1007/978-3-030-97546-3_41

Paper F (2) *Journal Paper:* R. O. Omslandseter, L. Jiao, X. Zhang, A. Yazidi, and J. B. Oommen, "The Hierarchical *Discrete* Pursuit Learning Automaton: A Novel Scheme With Fast Convergence and Epsilon-Optimality," *IEEE Transactions on Neural Networks and Learning Systems*, Early Access, pp. 1–15, IEEE, December 2022.
https://doi.org/10.1109/TNNLS.2022.3226538

Paper G (1) *Conference Paper:* R. O. Omslandseter, L. Jiao, and J. B. Oommen, "Enhancing the Speed of Hierarchical Learning Automata by Ordering the Actions – A Pioneering Approach," *AI 2022: Advances in Artificial Intelligence, AI 2022 (AJCAI 2022)*, Lecture Notes in Computer Science, vol 13728, pp. 775–788, Springer International Publishing, December 2022.
https://doi.org/10.1007/978-3-031-22695-3_54

Paper H (2) *Journal Paper:* R. O. Omslandseter, L. Jiao, and J. B. Oommen, "Pioneering Approaches for Enhancing the Speed of Hierarchical LA by Ordering the Actions," *Information Sciences*, vol 647, pp. 119487, Elsevier, November 2023.
https://doi.org/10.1016/j.ins.2023.119487

Paper I (1) *Conference Paper:* R. O. Omslandseter, L. Jiao, Y. Liu, and J. B. Oommen, "User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution," *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices, IEA/AIE 2020*, vol 12144, pp. 299–311, Springer International Publishing, September 2020.
https://doi.org/10.1007/978-3-030-55789-8_27

Paper J (1) *Journal Paper:* R. O. Omslandseter, L. Jiao, Y. Liu, and J. B. Oommen, "User Grouping and Power Allocation in NOMA Systems: A Novel Semi-Supervised Reinforcement Learning-Based Solution," *Pattern Analysis and Applications*, vol 26, pp.1–17, Springer London, July 2022.
https://doi.org/10.1007/s10044-022-01091-2

Papers Not Included in the Dissertation

Paper 11 (1) *Conference Paper:* R. O. Omslandseter, L. Jiao, and M. A. Haglund, "Field Measurements and Parameter Calibrations of Propagation Model for Digital Audio Broadcasting in Norway," *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall),* pp. 1–6, IEEE, August 2018.
https://doi.org/10.1109/VTCFall.2018.8690746

Contents

List of Figures

List of Tables

Part I

Summary of Contributions

Chapter 1

Introduction

We live in a century when computers, digital communication, and digital signal processing are becoming increasingly important in our daily lives. As more and more digital devices interact with each other, the requirements for rapid information exchange become more stringent, and increasingly more data, in various application domains, are collected in ever-increasing quantities. In this context, one of the fundamental questions encountered is that of knowing how we can better arrange and store the data in the increasingly-digitized world, and how we can obtain useful insights based on the enormous amounts of data collected. Understandably, due to the stochastic nature of the real world, changes may occur rapidly, and it is prudent to model and study the systems as being stochastic rather than static. To handle the stochastic nature, we utilize the field of Learning Automata (LA)[1], although not a new phenomenon in itself, as a vital component to aid us in these challenging tasks due to their extraordinary learning ability in dynamic environments. To efficiently utilize the algorithms within this intriguing domain, modifications and improvements are desirable so as to obtain, possibly, superior suitability and performance.

The research in this Ph. D. study was within LA, and is two-pronged. The first prong concerned the Fixed Structure Stochastic Automata (FSSA) type of LA. More specifically, it is concerned with solving partitioning problems in stochastic Environments via LA, demonstrated in an application. The second prong is concerned with improving the performance of the state-of-the-art algorithms within the Variable Structure Stochastic Automata (VSSA) type of LA. Moreover, for the first prong, we used the LA-based Object Migration Automata (OMA) algorithms for solving stochastic partitioning problems. In addition, we proposed the use of OMA in a mobile radio communications application. Furthermore, we presented inventions and discoveries within the field of OMA, and proposed new approaches for the OMA algorithms to solve problems that could not be handled previously. For the second prong, we proposed a new algorithm for a very recently-introduced type of LA, proved its performance mathematically, and made subsequent advancements concerning how we could enhance the algorithm's performance. In the following paragraphs, we will first briefly discuss the field of LA. Then, we will present our

[1]The term LA is used interchangeably to denote both the field of Learning Automata and the Learning Automaton itself, depending on the context it appears in.

novel OMA-based algorithms for solving partitioning problems. After that, we continue with our novel algorithms proposed in the VSSA domain. Lastly, we discuss our proposed LA-based solutions to the mobile radio communication application.

A fundamental problem in large amounts of data is one of discovering connections and patterns between the data points. However, finding such patterns can be complex and difficult, especially in high-dimensional spaces, with manual inspection. A pertinent issue in pattern discovery involves finding groups of data points that belong together, and is referred to as a *partitioning problem*. Partitioning problems involve splitting abstract data points into sub-sets based on underlying or known criteria[2]. In the Literature, we refer to the data points as objects, and a partitioning problem is often referred to as an Object Partitioning Problem (OPP) [1]. Within FSSA, we concentrate on the paradigm of OMA, which has been legendary in solving and monitoring partitioning problems. The LA-based OMA algorithm and its variants represent effective solutions to stochastic grouping problems. However, some of these algorithms' peculiarities have rendered them to be less applicable to real-life situations because they have been only able to handle equally-sized groups, i.e., Equi-Partitioning Problems (EPPs). This particular problem was investigated in this work, and several new OMA algorithms have been studied and proposed.

In the next major field of study, we visit the domain of fundamental LA algorithms themselves. In the Literature, we have two main categories of LA, namely FSSA and VSSA, respectively. As mentioned above, the OMA is an FSSA type of machine. Many advancements over the years have increased the algorithms' efficiency and accuracy, as further elaborated later in this dissertation. However, one problem that VSSA experiences is the diminishing increase/decrease in probability when the number of actions becomes large. Consequently, to mitigate this issue, the Hierarchical Continuous Pursuit Automaton (HCPA) was proposed by the authors of [2]. However, because it had an impediment when the accuracy requirement became stringent, in this Ph. D. thesis, we investigated how we could improve the state of the art in this field, and proposed the Hierarchical Discrete Pursuit Automaton (HDPA). In addition, in a completely different research direction, we discovered that when the LA is organized hierarchically, the ordering of actions within the algorithm greatly influences the machine's performance. For this reason, we studied and proposed methods that improved the algorithm's speed even further. We believe that this enhancement represents the state of the art.

The applications of the OMA algorithms are numerous. In this thesis, we studied one concrete application domain, i.e., in wireless communication. In the case of wireless communications, we observe a highly-relevant partitioning problem. In particular, in the case of Non-Orthogonal Multiple Access (NOMA) systems, the grouping of the users in the system greatly influences its performance. A well-known issue in mobile radio communication is that the available frequencies is a sparse commodity. With NOMA, it is possible to group the users into the same Resource Block (RB), thereby utilizing the frequency space in a superior manner. To efficiently

[2]This could also involve dividing the set of attributes in a database, or the set of cities where the data resides, etc. The term partitioning should thus be considered as an abstract concept.

make the method work, the users that are grouped together need to match in terms of their signal characteristics. However, the users in a mobile communication system possess highly stochastic behaviors, "moving" around in the corresponding space and thus, making their signal characteristics subject to substantial and rapid changes. Therefore, we need an adaptive method for finding and monitoring the groups. The OMA algorithms are potential solutions to the grouping problem in NOMA systems and were investigated and proposed as part of this dissertation.

With this brief introduction, the reader should have a basic understanding of the context and the different problems involved with this Ph. D. study. But before we proceed, in order for the reader to better understand the fundamentals of LA, we submit below a brief introduction of the families of LA and briefly present their characteristics.

1.1 Families of LA

Michael Lvovitch Tsetlin initiated the field of LA in the 1960's [3]. Although Tsetlin did not use the term LA, he introduced the concept of "automaton theory" and modeled the automata using matrices. In subsequent years, the term "Learning Automaton" was used to describe different schemes, both deterministic and stochastic. The principles of LA have become tightly intertwined with the field of Reinforcement Learning (RL), and today, the phenomenon of "Learning Automata" has become a standard description in both the concept of learning in itself, and in the field of Machine Learning (ML).

The field of LA is based on the theory of learning, realized in artificial units using complex computer programs. In LA, we have a learning unit (often referred to as the Learning Agent or LA) and a teaching unit (traditionally referred to as the Environment). The field of LA is rooted in psychology. Just as a child learns the correct actions from its parent through communication and corrections, the LA learns through interactions with the Environment. We can explain the LA operation by an action-feedback loop, where the agent selects an action from within its set of possible actions. Thereafter, the action is proposed to the Environment, which gives a response back to the LA. The feedback to the LA can be of different types, but generally, the agent will, based on the input from the Environment, update its behavior and subsequently, hopefully, learn the optimal action within its paradigm.

There are many variants of LA, and generally, they are sorted into two categories: FSSA and VSSA [4]. The first LA, introduced by Tsetlin, is an FSSA scheme. In the FSSA schemes, the learning is achieved through the LA by maintaining a "memory". The memory is realized through the use of *states*, and the machine's current state determines the current action. Learning is achieved by the LA by traversing its state space based on the feedback provided by the Environment. For the VSSA type, learning is achieved by the machine by maintaining an action probability vector, where the current action of the machine is determined by sampling this vector. Learning in VSSA is achieved through updating the action selection probabilities

based on the feedback of the Environment. Within both FSSA and VSSA, there have been advancements over the years. One of these advancements was the Estimator LA [5], and another introduced structure and a hierarchical operation in the LA [2, 6]. For the reader to obtain a basic fundamental understanding of the material in this dissertation, the aforementioned concepts are further elaborated on in the following subsections.

1.1.1 FSSA

In the family of FSSA, the functionality and the *memory* of the machine are maintained in *states*, identical to those possessed by Finite State Machines (FSMs) and flip flops [4]. The LA has a set of states, and the current state of the machine decides the action that the LA outputs at present. The LA traverses its states according to interactions with the Environment. There are often many states representing each of the actions. Thus, if the machine selects an action, and that action leads to a positive feedback from the Environment, the LA might reinforce the behavior of choosing that action by going deeper into the state space of that action.

The FSSA schemes are time-invariant and have a fixed policy for inter-state transitions and their behavior upon feedback from the Environment, and the action selection [4]. The FSSA schemes can be modeled as ergodic Marcov chains. Examples of FSSA machines are the Tsetlin, Krinsky, and Krylov LA [3]. Each of these machines has its own strategy of traversing the states and interacting with the Environment. Correspondingly, different FSSA types have distinctive convergence characteristics.

1.1.2 VSSA

VSSA represent another family of LA, and in this type of LA, the machine is characterized by its functionality and *memory* being maintained and connected to an action probability vector [4]. More specifically, in VSSA, the action that the LA selects is based on the action probability vector, $P(n)$, at time instant n, and the vector can be updated so that it has different action selection probabilities for $P(n+1)$. While transitions in FSSA are constant, they can change with n for VSSA. The action that is chosen is submitted as the input to the Environment, and based on the feedback from the Environment, the action probability vector will be updated. The updating algorithm can be varied and includes, among others, the Linear Reward-Penalty (L_{R-P}) scheme, the Linear Reward-Inaction (L_{R-I}) scheme, the Linear Inaction-Penalty (L_{I-P}) scheme, and the Linear Reward-ϵPenalty ($L_{R-\epsilon P}$) [4, 7]. In these different schemes, the probability vector is updated linearly. The updating can also be done in a non-linear manner [4, 7, 8].

1.1.3 Estimator LA

Pioneered by Thathachar and Sastry are the *Estimator-based Algorithms (EAs)* [9]. These machines have the fastest convergence to date, when they are considered as

a single entity, but they can also be part of a hierarchical structure, as addressed later [5]. EAs have a noticeably different paradigm, because they maintain running estimates of the reward probabilities, which they also utilize in the learning process. Such machines are characterized by phases of "exploration" and "exploitation", and the machine switches between these continuously. While updating the action probability vector, an "estimator" vector is also updated and maintained. This leads to faster convergence, because these estimates can be utilized in the learning of the machine so as to choose the superior actions more often.

Within EAs, we also include the concept of Pursuit Algorithms (PAs), where the learning algorithm increases the probability of the currently-recorded best action, and not of the action that is chosen. More specifically, the currently-best estimated action is *pursued*. The pioneering PA, followed an L_{R-I} continuous scheme, the Continuous PA (CPA) [9]. In [10], the authors introduced the Discrete PA (DPA), and in [11], the functionality of PAs was extended to Reward–Penalty (RP) paradigms as well. The pursuit-based LA methods, have been shown to be faster than traditional VSSA schemes [12, 13, 14, 15]. Also, the OMA-based Pursuit OMA (POMA) [16] and Pursuit Enhanced OMA (PEOMA) [17, 18] have incorporated the concept of pursuit into the OMA family.

1.1.4 Hierarchical LA

For the above-mentioned VSSA, the convergence becomes challenging when the number of possible actions is large. Understandably, for VSSA, the action probability vector has a dimension of R (for R actions), and its elements sum up to unity. When R is large, many of the action probabilities can have very small values and may not even be chosen, thus rendering the principle behind VSSA to be void. Already in 1981, the authors in [4] discussed the possibility of hierarchical structures in LA. Thathachar and Ramakrishnan followed up with the paper [6], where they demonstrated by a complexity analysis that the highest gain in computational savings happened when the number of actions in each LA in their hierarchical example system had two or three actions. Similar hierarchies were studied in [19] and [20]. A newer discovery was the more recent HCPA, and it constitutes the state of the art [2], where with a hierarchically structured binary tree consisting of PAs, the HCPA beat the earlier variants when the number of actions was high.

1.1.5 Brief Overview of Applications

The applications of LA and stochastic learning are manifold, and many examples are listed in the following books [4, 7, 21], including game-like problems and pattern classification. In [22], LA were used for network call admission. In relation to network communications, they were also used in [23] for traffic control and Quality of Service (QoS) routing [24]. LA have been used to adapt back-propagation algorithm parameters in feed-forward neural networks [25]. In [26, 27], LA-based methodologies were used for intelligent vehicle control. Additionally, they have been used for

determining roads in aerial images [28]. LA also represent a powerful solution for analyzing the stochastic model in wireless communications [29]. Understandably, LA are intertwined into a variety of different domains. The above-mentioned catalog of applications is only a brief summary of the many examples of applications in which LA represent compelling solutions.

Within the paradigm of OMA, we also find numerous applications. In [30], the OMA was used for cryptanalysis, where a substitution cipher was solved using only the plaintext and its analogous ciphertext. In image analysis, for finding conceptually similar images, the OMA demonstrated promising results in [31]. For finding the trustworthiness of a reputation system, the OMA was employed in [32]. The OMA has also shown that it was a highly effective solution for the problem of distributing traffic across multiple virtual computing instances in [33] and [34], demonstrating a 90 % cost reduction compared with other existing state-of-the-art approaches at that time. Additionally, the OMA was used for resolving queries in a distributed database with graph connections in [35]. The OMA has also been used to secure statistical databases in [36] and [37], beating the state-of-the-art algorithms for a Micro-Aggregation Problem. In [38], the OMA was used for outlier detection in the issue of noisy sensors in sensor networks, and in [39], it was used in an adaptive data structure solution to the paradigm of Singly-Linked Lists (SSLs) on SSLs, with outer and sub-list contexts.

In addition to all the applications mentioned above, more complex "constellations" of LA have also been introduced. The LA can be organized in complex cooperative learning schemes such as the newly introduced Tsetlin machine (TM) [40]. The TM is based on teams of LA working together in conjunctive clauses with rule schemes of propositional logic. The TM has shown competitive results compared with deep neural networks for various applications. As an example, the TM has been used for text classification in [41], novelty detection [42], semantic relation analysis [43], sentiment analysis in [44], and word sense disambiguation [45]. A formal mathematical analysis of the TM has been reported in [46] and [47], respectively.

1.2 Research Challenges within LA

Within the large field of LA, there are many research challenges and phenomena that can be investigated further to enhance the state-of-the-art algorithms and to improve the performance metrics of the current machines. In this regard, within this Ph. D. study, the research has been divided into one part concerning FSSA schemes and another part relating to VSSA schemes.

Within the FSSA schemes, the main research challenge has involved solving partitioning problems using LA. In addition, we have investigated a practical application for the FSSA algorithms in solving partitioning problems so as to demonstrate their power in solving and monitoring highly stochastic systems.

Regarding the VSSA schemes, within this Ph. D. study, we have mainly focused on two aspects concerning developing new algorithms, and in beating the state-of-the-art machines. The first aspect involves the problems that arise in VSSA when

we have a *large* number of actions, and the other aspect concerns incorporating the concept of *ordering* in the actions. In Figure 1.1, the research challenges are linked with the different papers that constitute the building blocks of this thesis.

1.2.1 Non-Equal Partition Sizes

Partitioning problems concern dividing a set of objects into specified (or unspecified) sub-sets based on specific known/unknown criteria. As mentioned earlier, in the Literature, such issues are often referred to as OPPs [1]. The splitting criterion for such problems can have various complexities, and render the problems to be NP-hard. The OPPs are similar to clustering problems but deal with a broader range of situations. OPPs do not necessarily group objects based on a measurable difference and can, for example, be based on abstract criteria such as the accessibility of objects together. An example of an OPP can be that of distributing different files in a geographically distributed database based on the accessibility of the users, or dividing users into positive and negative contributors based on their individual feedback, and also that of distributing power budgets among a set of users.

The LA-based FSSA, namely the OMA algorithms, can solve OPPs, and their great strength is that they can handle such problems in highly stochastic Environments [38]. However, the current OMA algorithms can only solve EPPs. More specifically, the existing OMA types can only handle grouping problems where the partitions are of equal size. Understandably, only being able to handle partitioning problems with equally-sized partitions is a limitation to the algorithms' applicability to real-life situations. The first OMA variant was introduced more than 35 years ago, and until now, enabling the OMA algorithms to handle Non-Equi-Partitioning Problems (NEPPs), where the partitioning problem can have non-equal partition sizes, has remained unresolved. Therefore, one of the main research challenges within the OMA paradigm involves expanding the range of problems that the algorithms can be applied to by making them able to handle both EPPs and NEPPs. We shall deal with these problems quite comprehensively.

1.2.2 Problems with Large Number of Actions

In VSSA, as explained before, the action selection probability is maintained in a vector, and the learning mechanism operates in a multiplicative manner for the continuous-type VSSA and with specific increase/decrease strategies for the discrete-type VSSA. The corresponding action selection probability is normally initialized as $\frac{1}{R}$ per action, where R is the number of actions in the LA. Consequently, when R becomes larger, the probability of a particular action initially being chosen decreases. Likewise, for the updating mechanisms, whether for the continuous or discrete type VSSA, the LA requires a smaller learning parameter to guarantee actual learning. Thus, the LA should not just instantly increase one of the action's probability so much that it immediately becomes the only action that the machine selects. Although we can tune the learning parameter, the problem can still be that the LA

does not sufficiently explore all the actions, and we might even have the case that the LA does not choose some of the actions at all!

The ϵ-optimality of absorbing CPA was presented in [48]. Similarly, the ϵ-optimality of DPA was presented in [49], and its finite-time behavior was formally proven in [50]. Generally, in all of these proofs, the analysis involves dealing with a small-enough learning parameter, to ensure that the algorithms ϵ-optimally converge to the optimal action. However, the smaller the learning parameter is, the longer it will take for the algorithm to converge. Thus, in a practical system, we might not want to wait for such a long duration, and certainly not for an "infinite" time. Therefore, the problem of handling a large number of actions in LA is a complicated one, and a prominent research challenge.

As earlier mentioned, the hierarchical LA constitute efficient solutions to problems where the number of actions is large. The recent discovery of the HCPA presented in [2], utilizing "pursuit"-based LA in the hierarchy (PAs), represents a quantum step in rendering the LA to be able to solve problems with a large number of actions very efficiently. The HCPA is the state-of-the-art machine in VSSA. However, there are still challenges that remain unresolved with the HCPA. In more detail, the HCPA demonstrates a more sluggish convergence as any action selection probability approaches unity, especially for high accuracy requirements. Therefore, one important research challenge is that of solving these issues, and possibly further enhancing the state of the art within this field.

1.2.3 Incorporating Ordering in the Actions

Over the decades, ever since Tsetlin's introduction, researchers have proposed new concepts and fundamental principles to advance the LA's accuracy and speed. The advancements include utilizing probability updating functions, discretizing the probability space, and using the "Pursuit" concept. Recently, the HCPA phenomenon that incorporates structure and a hierarchical ordering of the actions, has significantly improved both the accuracy and speed when the number of actions is large [2]. Consequently, passing on the legacy so that constant improvements see the light of day is essential for LA's development and applicability to real-life problems.

Although the advancements leading up to the field of LA that we have today are manifold, there are still problems that remain. With the HCPA, the actions are located at the leaf of a tree consisting of LAs. These actions sit at the terminal nodes of different paths through the tree, and one might start to wonder whether the *ordering* of the actions at the leaf level influences the machine's performance. As addressed in 1984 [51], such structures can have impediments because the paths through the tree are not always optimal. However, the solution proposed in the paper does not consider the more recent discoveries and will, likewise, not work with them. Linked to the concept of Random Races [52], an interesting field of study is whether the ordering of the actions can be incorporated with the current machines. This is one avenue where we have done some of our research.

1.2.4 Practical Applications

As the field of ML increases, so do the requirements for the algorithms' applicability to real-life scenarios. Numerous areas and domains are crucial and interesting, and it would be beneficial if they can be tackled using the various tools for analysis that ML provide. One of these fields is mobile radio communications, which is within the focus of this thesis.

Mobile radio communication is a well-established field of research, and a promising technique is NOMA [53, 54]. In NOMA systems, users can be multiplexed into the same RB, which utilizes the frequency bands more efficiently. In these systems, the users that are grouped need to be carefully selected, and their power levels need to be adjusted accordingly. In terms of finding grouping solutions, the study is still in its infancy. In [55], a dynamic method was investigated, where the channels were sorted from high to low and grouped on this basis. In [56], a K-Means clustering scheme was employed to group users based on geolocation in, e.g., school halls. Maximum weight matching was used in [57] to build groups and allocate power. Proportional Fairness (PF) was applied through an exhaustive search in [58], and also such an approach was applied in [59]. In [54], they analyzed NOMA uplink systems, looking at the performance improvement if the users were paired optimally given sub-optimal power allocation schemes. Here they showed that the optimal user pairing can be found by utilizing the Hungarian algorithm in the configuration of a multi-antenna Base Station (BS) with single-antenna users. However, the methods proposed commonly assume, e.g., a particular channel fading. In previous solutions, although channel fading was assumed to be following a certain random distribution, user grouping and power allocation were traditionally carried out based on a *known* instantaneous sample from the distribution. However, due to the stochastic nature in mobile communications, e.g., the mobility, the channel conditions, may be changed significantly over time even for slow fading. Therefore, the practical utilization and solutions to the problem remain an unresearched area. Consequently, by investigating more practical approaches, considering the actual channel is an interesting research topic that could enhance the field of NOMA. This is one avenue where we have utilized LA in an applied manner.

1.3 Research Objectives and Methodology

As highlighted above, substantial research effort has been made in the paradigm of LA, both in terms of the development and analysis of the algorithms, and their practical applications. In spite of this, there are still aspects and issues that have not yet been considered in the Literature. For instance, in the case of OMA, the algorithms are not able to handle NEPPs. Additionally, within partitioning, detecting patterns and monitoring them over time is a complex issue, and many interesting and prominent applications have yet not been investigated. For example, the grouping and monitoring of the users in mobile radio communications systems is one field where LA can provide advantageous solutions that can handle stochastic behavior

within the domains. Further advancements within VSSA are also crucial to continue the application of these methods, increasing their accuracy and efficiency so that they can be utilized to a greater extent in real-world applications, and trying to beat state-of-the-art algorithms within the field. The open research questions and behaviors in the field of LA that can be investigated further are manifold. Thus, in this Ph. D. study, we attempted to answer the following research questions:

- **Question 1:** The algorithms within the OMA paradigm can reveal hidden and unknown objects belonging together based on information about their joint accessibility, i.e., the so-called *queries*. The basis of these queries can be modeled in numerous ways. Suppose we wanted to find the optimal placement of files in a geographically distributed database. In that case, the OMA could monitor the queries of the different users accessing the files in the system, and would thereafter be able to find an optimal placement of these files. In such a system, we might have no prior information on which to base the groupings. Thus, we only have information about the files being accessed over time. The OMA algorithms can solve such problems and even handle them when there are high degrees of uncertainty and stochastic behaviors. However, the existing OMA algorithms can only handle grouping problems where the partitions have equal sizes. Since we do not always have equally-sized groups, the first research question was as follows: *Can we develop OMA algorithms that can handle non-equally sized groups?*

- **Question 2:** In VSSA, the convergence rate and accuracy definitely degrade as the number of actions increases. The HCPA was introduced so that LA could handle a higher number of actions efficiently, and this method constitutes the state of the art. Although the authors of [2] demonstrated and mathematically proved the superior performance of this machine compared to earlier existing variants, the HCPA still has an impediment when the accuracy requirement of the algorithm becomes high (e.g. above 0.99). Consequently, the second research question of this study became: *Can we improve the state of the art in LA so that the performance is not degraded even if the accuracy requirement of the algorithm is high and is closer to unity?*

- **Question 3:** In the hierarchical structures of LA, the actual actions selected depend on the specific path through the tree structure. There can be multiple actions per LA in the tree, and the policies within the tree can be different depending on the LA type and the organization of the tree. The actions selected by the machine are traditionally located at the leaf level of the tree. Consequently, for a specific action to be chosen, all of the branches leading to that action must be activated. When the actions depend on a hierarchy, like in the case of a hierarchical tree structure of LAs, we can experience problems that are not present within traditional LA, where all of the actions possess an equal play. Indeed, the placement of the actions within the tree structure seems to impact the algorithm's performance. Therefore, the research question

becomes: *Can we find out why the placement of the actions in hierarchical LA structures has an impact on the algorithms' performance, and can we propose a solution to mitigate this issue in practical situations?*

- **Question 4:** In mobile radio communications, the available frequency bands are a sparse resource. Consequently, developing methods that utilize the frequencies more effectively, are crucial. The NOMA system is a promising technique in mobile radio communications because it allows multiple users to utilize the same frequency bands efficiently. However, we must favorably group the users for this technique to function adequately. Additionally, mobile users are extremely stochastic in their behavior, complicating the issue. Current solutions to the grouping in NOMA systems are mathematically bounded and restricted, making them difficult to be implemented in practice. The research question here, therefore, was: *Can OMA algorithms find user groupings in NOMA systems in an improved manner and still handle the problem's stochastic nature?*

The research questions listed above are organized in a logical order. However, although the research questions are related to some extent, they are rather standalone in terms of the order in which they have to be solved. The first question is related to the field of FSSA and relates to the algorithms within the OMA family. The second and third research questions are associated with the development of VSSA schemes and relate to enhancing the state of the art and improving hierarchical structures within LA. The fourth research question is related to practical applications, and concern using the OMA for solving grouping-related problems in wireless communications.

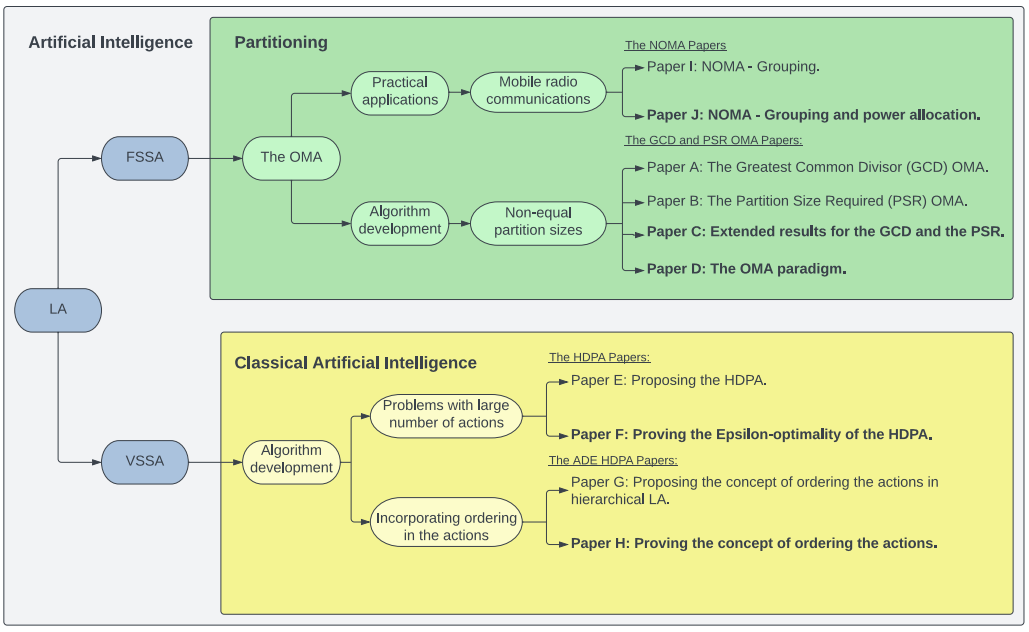


Figure 1.1: The connections between the research topics and the papers included in this dissertation.

In Figure 1.1, we depict the research work and connection between the research areas. The figure also shows the different papers included in the dissertation and specifies the topics they belong to. As depicted in the figure, we explore our research in two main directions, i.e., the FSSA and VSSA direction. In the FSSA case, we investigated the OMA paradigm, with the goal of developing algorithms that extend the OMA's functionality so that it can handle both EPPs and NEPPs. Additionally, we demonstrated the applicability of the OMA to a practical application, namely the field of NOMA in communication system. Thus, to address the research questions listed above, we identify the following six research goals, and they are achieved through the scientific contributions of the thesis highlighted in Papers A-J:

- **Goal 1:** Propose a novel OMA-based scheme that relaxes the equi-partition size group constraint that the algorithms currently demand. The aim of the first goal was not to entirely remove the partitioning size constraint, but to improve the algorithms so that they could offer more flexibility in group size configurations in the current partitioning problems (Paper A, C, and D).
- **Goal 2:** Propose a novel OMA-based method that allows the OMA to be able to solve EPPs and NEPPs efficiently. Additionally, we should be able to use the approach for all the existing algorithms (Paper B, C and D).
- **Goal 3:** Propose a novel scheme within the field of VSSA that beats the state of the art and, additionally, addresses the issue of rather sluggish convergence for high accuracy requirements in hierarchical LA (Paper E).
- **Goal 4:** Formally prove, the ϵ-optimality related to the convergence of the proposed scheme addressed in the goal listed above (Paper F).
- **Goal 5:** Investigate and propose a solution to the problem of improving the performance in hierarchical schemes by optimally determining the the placement of the actions within such structures (Paper G).
- **Goal 6:** Formally prove the reason for the varying convergence speeds based on the placement of the actions in hierarchical LA structures (Paper H).
- **Goal 7:** Demonstrate the use of OMA for grouping and monitoring the stochastic moving mobile users in a NOMA system (Paper I).
- **Goal 8:** In the context of Goal 7, we additionally propose detailed solutions for power allocation in such systems (Paper J).

The approach for addressing the research questions and achieving the goals of this Ph. D. study includes mathematical modeling and computer simulations. In the papers presented, in some cases, we demonstrate the performance and convergence of some of the methods by formal mathematical analyses. In all of these contributions, we demonstrated the performance of different proposed schemes through computer simulations. The simulations were mainly conducted using Python programming and related IDEs (like Spyder and Pycharm). For simulating mobile radio communication channels, MATLAB became a crucial tool.

1.4 Organization of the Dissertation

This dissertation is composed of two parts. Part I summarizes the main discoveries and research contributions of this Ph. D. study. Part II contains a collection of papers that represent and highlight the main research contributions. The organization of the papers are presented in Figure 1.1, and the overview is connected to the outline of the research challenges presented in Section 1.2. The papers that are marked in bold are journal papers.

The remaining chapters of Part I is summarized as follows:

Chapter 2: Chapter 2 will briefly survey the theory of FSSA, and introduce the reader to this fascinating field. Additionally, the estimator, pursuit, and transitivity concepts of the different historical enhancements to the OMA will be detailed concerning partitioning problems. In this way, the reader will have a concise introduction to the field. Specifically, we first present the existing algorithms and their related aspects before we present the algorithms proposed in this dissertation in Chapter 4. We will summarize Chapter 2 before the VSSA part of LA is presented in Chapter 3.

Chapter 3: Chapter 3 will consider the VSSA part of LA. First, we will present the concept of the VSSA, the Estimator schemes, the Hierarchical LA, and their applications in more detail. This brief introduction will equip the reader with the background theory needed for comprehending the contributions within the field of VSSA given here. After that, we will examine the motivation for developing faster and better algorithms, and present the state-of-the-art algorithms and approaches. This chapter is also summarized, before we continue with Chapter 4.

Chapter 4: Chapter 4 will consider our novel contributions to the FSSA field of LA. More specifically, we explain the motivation for developing new OMA schemes, and we subsequently explain these innovations in more detail. We close the chapter with an overall summary, before we proceed with the next chapter about our novel contributions to the VSSA field in Chapter 5.

Chapter 5: Chapter 5 presents our proposed algorithms in the VSSA field of LA. Thus, we first present a summary of the HDPA, before we explain the concept of ordering of the actions in hierarchical LA structures, and our ordering solution. We briefly summarize the chapter in an overall manner, before we proceed with the application study of this Ph. D. work in Chapter 6.

Chapter 6: Chapter 6 presents the application-based contribution of this Ph. D. research. The application is related to grouping issues, and we utilized the FSSA-based OMA methods to solve it. We first explain the central concepts of the application, followed by the proposed solutions separately and systematically. The chapter is briefly summarized, before we proceed with a more detailed and overall conclusion of the dissertation in Chapter 7.

Chapter 7: Chapter 7 presents the conclusion of the dissertation, and we discuss some modifications and behaviors that can be considered in future work. The dissertation has two main parts regarding the research contributions, but they are considered collectively in this overall closure of the dissertation. After that, we proceed with Part II, where we systematically present the various research papers.

Chapter 2

Fixed Structure Stochastic Automata (FSSA)

The study of behavior, statistical decision-making based on prior experiences, the solution to the two-armed bandit problem, and the theory of rational decision-making in stochastic Environments, are all components of LA [4]. LA has its origins in the field of psychology and can be viewed as a combination of these aspects. Intriguingly, we can use human behavior as a source of inspiration for creating and guiding LA entities, and in some cases, it can also help us understand and explain their behavior better. This is a fascinating feature of LA, as it allows us to draw connections between human decision-making processes and the ways in which LA agents operate and learn. Thus, the research area is both interesting in terms of making ML algorithms that are better and more versatile, and in the fundamental aspect of enhancing the AI algorithms' human-ness in decisions and behavior.

A LA can be defined as an *Agent* that makes decisions while operating in an Environment that is random or uncertain. These non-human agents are implemented through artificial, complex computer programs, where they are able to make decisions and adapt to the world they operate in through interactions with a so-called *Environment*. They *learn* the behaviors that, traditionally, ensure they have the highest probability of being rewarded for their actions. Thus, the LA operation can be modeled as an action-feedback loop, where the LA selects an action (makes a decision), and the Environment that it operates in gives feedback that makes the agent change its behavior over time.

As mentioned earlier, we have two main variants of LA, FSSA and VSSA, respectively. In this chapter, we will focus on the FSSA variants. The original LA, introduced by Michael Lvovitch Tsetlin in the 1960s, was an FSSA. These LA have structures that do not change with time, and their current state determines their current behavior. An FSSA can be modeled as a finite state machine, with a set of states, input events, and transitions between the states [4]. Consequently, the behavior of an FSSA is determined by the number of states that it has, what the states correspond to, and how the machine traverses these states. In the first part of this chapter, we will explain the main characteristics of FSSA, including a detailed example of their operation. The OMA algorithms are FSSA schemes. Consequently, the

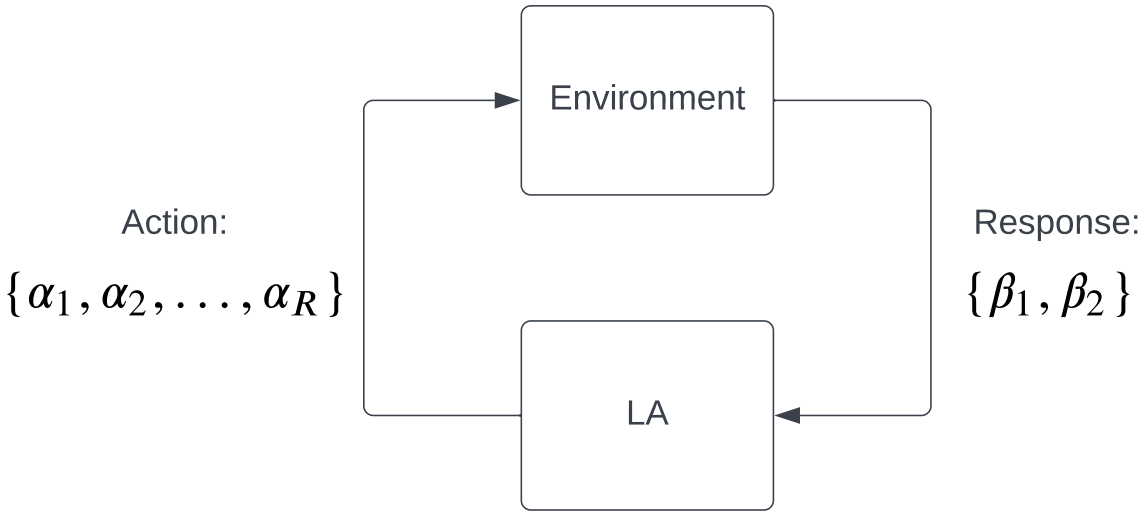


Figure 2.1: A visualization of the LA-Environment model.

second part of this chapter concerns how such techniques can be used for grouping problems, with a detailed explanation of the different OMA schemes.

2.1 The Family of FSSA

The automaton approach to achieve learning involves determining the optimal action from among a set of actions [4], which is done by the LA acting as a *learner* and the Environment acting as a *teacher*. As briefly mentioned above, we can model the operation of the LA and the interaction with the Environment as a feedback loop between them, as depicted in Figure 2.1. The LA selects an action (α_i) from its R possible actions. The selected action is presented as the input to the Environment, and the Environment responds with a feedback. The feedback can be either discrete or continuous, but it is commonly (as in this thesis), binary, where the Environment either responds in a positive manner with a *Reward* or in a negative manner with a *Penalty*, according to the input action.

In order to better understand the characteristics of LA, we need some mathematical precision. To do this, we follow notations similar to the ones established in [4][1]. In mathematical terms, the Environment is composed of three components: $\mathcal{A}$, $\mathcal{C}$ and $\mathcal{B}$, where $\mathcal{A} = \{\alpha_1, \alpha_2, ..., \alpha_R\}$ is the set of R possible actions that the LA can choose from. Unknown in real life, but possibly known in simulation Environments, are the probabilities of the Environment responding with a Penalty for the possible actions, denoted by $\mathcal{C}$, where we have a single probability for each action. Therefore, $\mathcal{C} = \{c_1, c_2, ..., c_R\}$, where each element, c_i, is the probability of the Environment's response being a Penalty for the given action, α_i. The output from the Environment (which depends on $\mathcal{C}$), i.e., the Environment's response to a certain action, is denoted by $\mathcal{B}$. As mentioned earlier, the most common feedback is from a

[1] In the interest of brevity, we merely present here the minimal knowledge required for the reader to understand the concepts presented within this dissertation. A more thorough explanation can be found in [4].

binary output set, where $\mathcal{B} \in \{\beta_1, \beta_2\}$, and $\beta_1 = 0$ might correspond to a Reward, and $\beta_2 = 1$ might correspond to a Penalty.

The reader should remember that in a LA system, the interactions of the LA with the Environment are often considered in a temporal space that is discrete. These discrete intervals can have a relation to time, e.g., an action is performed every minute. However, in order to generalize this phenomenon, we denote the discretized steps by an index "n". As a result, the LA selects an action for each n, and consequently receives an input (feedback) from the Environment. It is important to keep in mind that the probabilities in $\mathcal{C}$ could change throughout the operation depending on the system that is being represented, in which case it is referred to as a "non-stationary" Environment. On the other hand, the Environment is said to be "stationary" when the penalty probabilities do not vary with time.

The LA, on the other hand, can be formalized by the following five parameters:

$$\Theta = \{\mathcal{S}, \mathcal{A}, \mathcal{B}, \mathcal{F}(.,.), \mathcal{H}(.,.)\}, \tag{2.1}$$

where $\mathcal{S}$ is the set of states and $\mathcal{S} = \{1, 2, ..., RS\}$. Thus, there are RS states in total, with S states per action. As mentioned earlier, $\mathcal{A}$ is the set of possible actions of the automaton, and $\mathcal{B}$ represents the feedback that the automaton can receive from the Environment. Consequently, the selected action corresponds to the *output* of the machine, and the Environment's feedback corresponds to the *input* to the machine. Furthermore, $\mathcal{F}(.,.)$ maps the machine's current state and input from the Environment into its next state, which can be formally described as $\mathcal{F}(.,.) : \mathcal{S} \times \mathcal{B} \to \mathcal{S}$. Depending on the LA's configuration, the machine's transition from one state to another, might change its next chosen output action, which can be the case in FSSA (or the probability of choosing the different actions, as in the case of VSSA, detailed later in this dissertation). To represent the change of the LA's chosen output action, we use the function $\mathcal{H}(.,.)$, as in [4]. Consequently, $\mathcal{H}(.,.)$ maps the current state and current input to the machine into its current output, expressed formally as $\mathcal{H}(.,.) : \mathcal{S} \times \mathcal{B} \to \mathcal{A}$. However, if the chosen output action depends only on the machine's current state, referred to as a state-output automaton, its operation is better represented by the output function $\mathcal{G}(.,.)$, expressed formally as $\mathcal{G} : \mathcal{S} \to \mathcal{A}$. The FSSA machines are state-output automatons, and they also have other distinct characteristics, discussed, in more detail, below.

2.1.1 The FSSA Characteristics

As mentioned above, the FSSA machines are state-output automata. In addition, $\mathcal{S}$, $\mathcal{B}$, and $\mathcal{A}$ are finite sets. More specifically, the concepts can be summarized as follows [4]:

- The states maintain the *memory* of the automaton, and at any time instant n, its state can be represented as $\phi(n)$, where $\phi(n) \in \{1, 2, ..., RS\}$. Thus, the current state of the machine is from a finite set of states.

- The output, i.e., the selected action, of the machine at any time instant n, is represented as $\alpha(n)$, and $\mathcal{A} = \{\alpha_1, \alpha_2, ..., \alpha_R\}$. Consequently, the action selected by the automaton is from a finite set of actions (and depends on the output function, as elaborated below).

- The input, i.e., the feedback from the Environment to the automaton is traditionally also from a finite set (note that it can also come from an infinite set [4]), where the feedback at any time instant is denoted as $\beta(n)$, and $\mathcal{B} = \{\beta_1, \beta_2, ..., \beta_m\}$, or $\mathcal{B} = \{a, b\}$ (where a and b are real numbers).

- The transition function is the operation that ultimately changes the behavior of the machine. This mapping considers the machine's current state and the input feedback to decide on the machine's state at the next time instant. Thus, $\phi(n+1) = \mathcal{F}[\phi(n), \beta(n)]$. This function can be either deterministic or stochastic.

- The output function is the operation that determines the action selected by the automaton at any time instant, n, as $\alpha(n) = \mathcal{G}[\phi(n)]$. This function can also be either deterministic or stochastic, and as also mentioned earlier, for FSSA, these automata are state-output machines. Thus, the actions determined as the outputs are based on the current state that the machine is in.

The information presented in the above list may be represented graphically as seen in Figure 2.2 [4]. As we can observe from the figure, the automaton has a state, which can be changed by the transition function, and this depends on the machine's current state and the input from the Environment. Because FSSAs are state-output automatons, the machine's current state determines the result of the output function, i.e., the chosen output action. Informally, the transition function handles what happens upon an unfavorable or favorable response from the Environment, and is responsible for the automaton eventually adjusting its behavior and *learning* the optimal action. The output function decides which action is chosen as the output based on the machine's current state.

Before we proceed, it is important to mention that if we have an automaton that for a given state and input, the next state and action are completely predetermined with no uncertainty, the automaton is said to be *"deterministic"* [4]. A deterministic automaton has both a deterministic transition and output function. However, if either of these functions are stochastic, meaning that they have some degree of uncertainty, the automaton is said to be *"stochastic"*. An FSSA, can be either deterministic or stochastic. However, in most cases, the LA has some type of uncertainty, e.g., in the case when we have more than two actions, and/or the transition from one action to another is done in a probabilistic manner, or when it incorporates the *Pursuit* concept (explained in more detail later). In addition to the characteristics above, FSSA[2] are known by their independence from n and the input, in itself, in

[2]In this dissertation, as in most of the Literature, we generalize the fixed structure automata, and refer to them as FSSA, although some schemes might, in reality, only exhibit deterministic behavior.

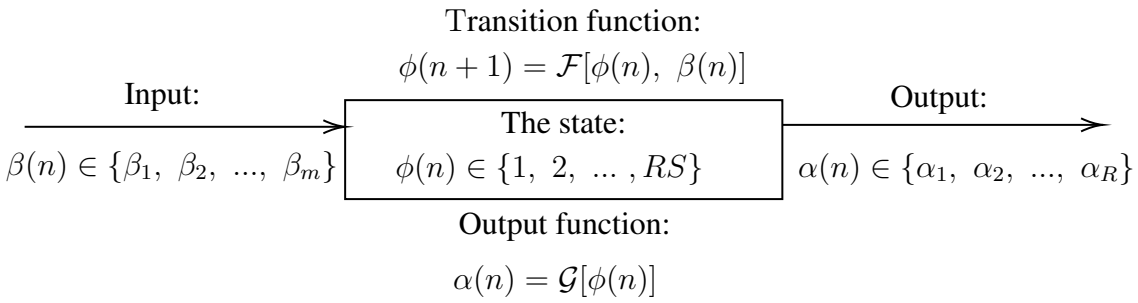


Figure 2.2: A generalized FSSA machine and its components.

terms of the transition function and the output function, respectively.

Another essential feature of stochastic automata is that they can be made deterministic by increasing the number of states inside the automaton. In this way, a stochastic automaton can be viewed as being deterministic, even if its transition or output function is stochastic. Because of this characterization of S, it is possible to conduct an analysis of the input-output behavior shown by the machine. This is due to the fact that the output function can be rendered deterministic. The formal evidence of this behavior has been excluded from this section, but it can be found in [4].

2.1.2 Examples of FSSA

Several distinct families of FSSA machines are characterized in the LA paradigm. The behavior of these automata can be altered in a wide variety of ways. In this part, we will cover two instances of such LA so that the reader may better understand the general idea behind these LA. More specifically, we will consider the Tsetlin Automaton and the Krylov Automaton.

2.1.2.1 The Tsetlin Automaton

The Tsetlin automaton is one of the simplest FSSA schemes. As for a state-output automaton, the current state that the automaton resides in determines the action it chooses as its output, i.e., the state determines the action that the automaton selects. The Tsetlin automaton has a pre-determined set of states that corresponds to each of the actions. As long as the automaton is in the set corresponding to an action, e.g., α_1, it will select α_1 for each n. These states are ordered such that upon a favorable response from the Environment (a Reward), the machine moves deeper into the state space of its current action. Upon receiving an unfavorable response from the Environment (a Penalty), the automaton moves shallower in the state space of its current action. More specifically, the Tsetlin automaton has S states per action. These states corresponds to the *memory* of the machine. Thus, S is the number of memory locations of the machine. Consequently, it requires a maximum of S consecutive unfavorable feedbacks from the Environment for the automaton to change to another action if it is currently inside an unfavorable action.

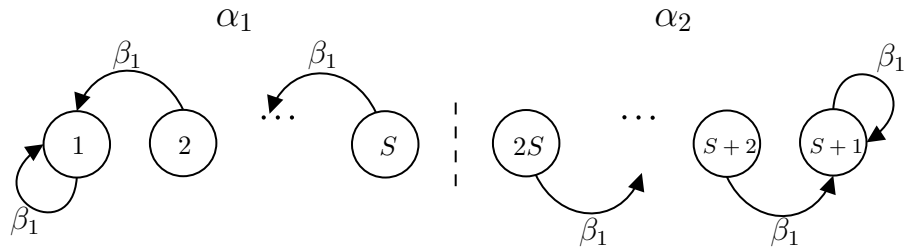


Figure 2.3: The Tsetlin automaton's operation upon a Reward.

In Figure 2.3, we visualize the functionality of the Tsetlin automaton upon receiving a Reward. As we can observe from the figure, we have two actions, α_1 and α_2, respectively. The states from 1 to S belongs to α_1, and the states from $S+1$ to $2S$ belongs to α_2. For example, if the automaton is currently in state 2, it chooses α_1. In this case, if the Environment responds with a Reward (β_1), the next state of the automaton is state 1. In other words, upon receiving a Reward, the behavior of selecting that action is reinforced. In this way, the automaton will *remember* this action longer, and it will require one additional Penalty to unlearn the current behavior. For a favorable response, the automaton will only move deeper into the state space, and it does not have the possibility of changing to another action.

In Figure 2.4, we see an example of the automaton's behavior upon receiving an unfavorable response from the Environment. In this figure, we can observe that the automaton will move shallower inside the state space of a certain action upon an unfavorable response, and change its action if it is currently in a so-called *boundary state*. The boundary states in Figure 2.4 are S and $2S$, respectively. Consequently, for R actions, we find the boundary states in rS, where $r \in \{1, 2, ..., R\}$. When the automaton is in a boundary state and receives an unfavorable response (a Penalty), it will switch to another action, as discussed below. If we consider the case that the automaton is in state 1, it outputs α_1, and receives a Penalty (β_2) from the Environment, the next state of the automaton is state 2. In this way, the automaton will eventually change to another action if its current action is unfavorable.

When the Tsetlin automaton has more than two actions, the number of actions that the automaton can switch to upon an unfavorable response is one less than the total number of actions, which is denoted by $R-1$. In the traditional Tsetlin automaton proposed by Tsetlin, this is done in a sequence, as depicted in Figure 2.5. As visualized in the figure, the automaton moves to the boundary state of α_2 when it is in the boundary state of α_1. Furthermore, by a simple extension, we have $\alpha_2 \rightarrow \alpha_3$, $\alpha_3 \rightarrow \alpha_1$, respectively at the boundary states. On the other hand, these transitions can also be done in a stochastic fashion, which is a more typical technique nowadays [4], making the Tsetlin automaton stochastic.

2.1.2.2 The Krylov Automaton

The behavior of the Krylov automaton is similar to the Tsetlin automaton upon receiving a favorable response from the Environment. Thus, the Krylov automaton follows the same operation, as depicted in Figure 2.3, upon a Reward [4]. Also

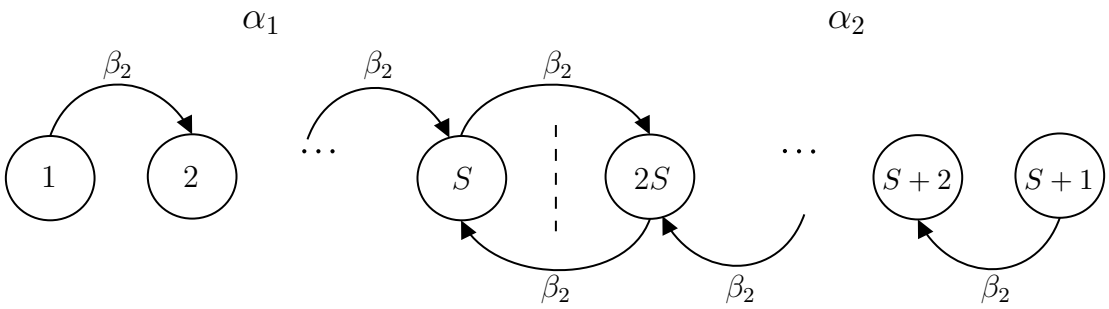


Figure 2.4: The Tsetlin automaton's operation upon a Penalty.

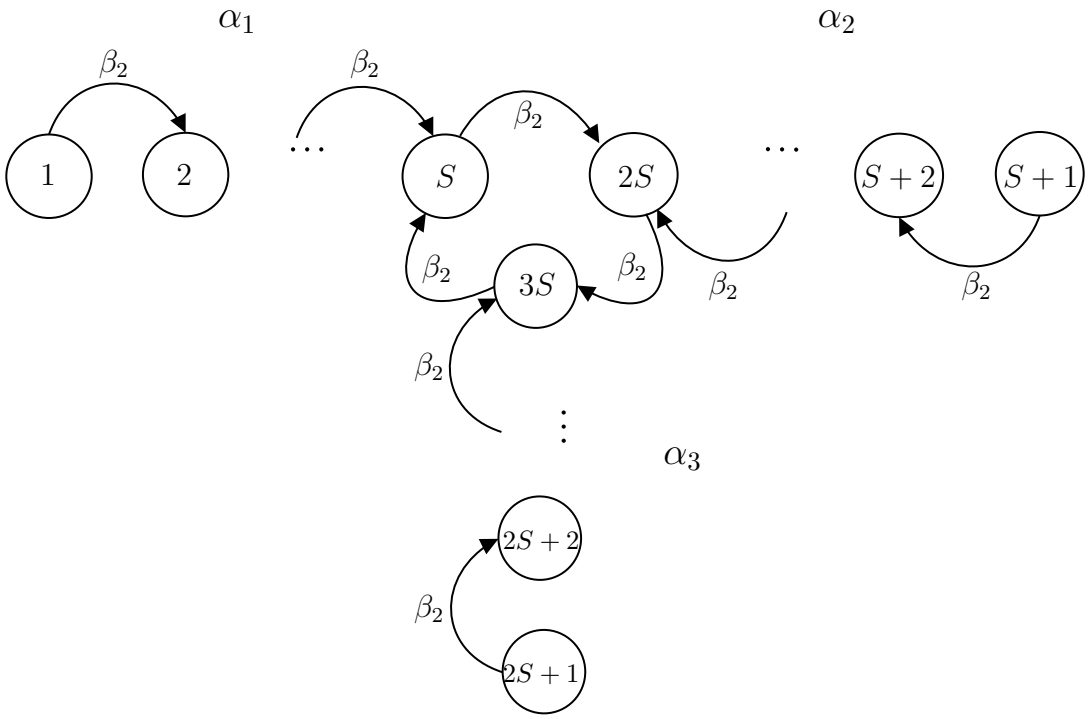


Figure 2.5: The Tsetlin automaton's operation upon a Penalty for three actions.

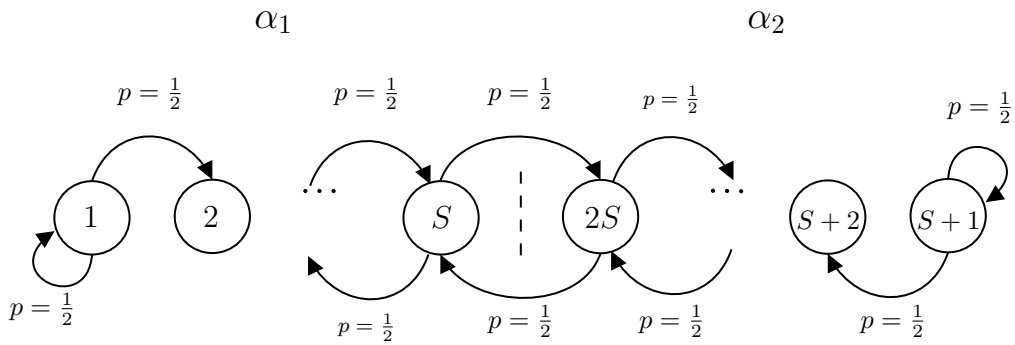


Figure 2.6: The Krylov automaton's behavior upon a Penalty.

similar to the Tsetlin Automaton, the Krylov automaton has a set of states per action, and the current state of the machine determines its output. The automaton traverses the states in accordance with the feedback from the Environment. The difference between the Tsetlin automaton and the Krylov automaton is in terms of the machines's operation upon receiving an unfavorable response from the Environment. The Krylov automaton has a sequential handling of switching between actions, like the Tsetlin automaton. However, as for the Tsetlin automaton, the switch of action can also be done in a stochastic manner.

In Figure 2.6, we visualize the operation of the Krylov automaton upon receiving a Penalty from the Environment. Upon a Penalty, the Krylov automaton has a stochastic behavior. Thus, the automaton either goes shallower (or switches action) or deeper in the state space of the action that it currently is in. More specifically, when the automaton is neither in the innermost or a boundary state, it will go deeper with probability $\frac{1}{2}$, or shallower with a probability $\frac{1}{2}$. If the automaton is in an innermost state, it will stay in its current state with probability $\frac{1}{2}$, or to the next shallower state with probability $\frac{1}{2}$. In a sense, the automaton randomly chooses to treat the Penalty as a Reward (as per Tsetlin's perspective) in all cases. This behavior can aid the automaton to remain more stable, even in highly stochastic Environments.

2.2 Assessment of LA Behavior

The optimal action for an automaton is the action that results in the minimum probability of receiving a Penalty from the Environment when it chooses that action [4]. In order to consider whether an automaton is able to *learn*, it needs to perform better than a "pure-chance automaton". Similar to the notation in [4], we denote the average penalty for a given action probability vector as $M(n) = \sum_{i=1}^{R} c_i p_i(n)$, where p_i is the probability of the automaton selecting action α_i at time n. For a "pure-chance automaton", $M(n)$ is a constant M_0 and $M_0 = \frac{1}{R}\sum_{i=1}^{R} c_i$.

Within the paradigm of LA, a machine that is better than a "pure-chance automaton" is said to be "expedient". The expedient behavior can be mathematically represented as:

$$\lim_{n\to\infty} E[M(n)] < M_0. \tag{2.2}$$

Another characterization for assessing the learning behavior is "optimality" [4]. More specifically, the automaton is said to be optimal if the following statement is true:

$$\lim_{n\to\infty} E[M(n)] = c_l, \tag{2.3}$$

where $c_l = \min\{c_i\}$. Thus, if an automaton is *optimal*, it will choose α_l asymptotically, with probability one. However, it is impossible for any LA to satisfy this criterion. We thus have the concept of ϵ-optimality. More specifically, an automaton is said to be ϵ-optimal if:

$$\lim_{n\to\infty} E[M(n)] < c_l + \epsilon, \tag{2.4}$$

where ϵ is an arbitrarily small value, and $0 < \epsilon$. In simpler terms, an ϵ-optimal automaton will, within an infinite timeframe, eventually converge to the optimal action with a probability that is arbitrarily close to unity. The reader should note that these assessment criteria relate to the FSSA and the VSSA types of LA in different manners, as highlighted below, and in the subsequent chapters.

Additionally, we also have the *absolutely expedient* behavior [4], which is an alternative condition to ϵ-optimality. An automaton is said to be absolutely expedient if:

$$\lim_{n\to\infty} E[M(n+1)|p(n)] < M(n), \tag{2.5}$$

and that this inequality is true for all n and $\mathcal{C}$, where $p_i(n) \in (0,1)$. More specifically, this means that the average penalty strictly decreases with n. In practical terms, it means that in the expected sense, the automaton learns monotonically at every time instant, eventually abandoning the unfavorable actions, and converging to the optimal action with an arbitrarily high probability.

Above, we have discussed the assessment criteria for the LA's behavior in a general manner. However, unlike the VSSA schemes, an FSSA scheme has a memory that is realized through the automaton's states. Let us, therefore, consider the asymptotic ϵ-optimality as the number of states goes to infinity [4]. More specifically, a deterministic automaton is said to be ϵ-optimal if there exist an S_1 such that $S > S_1$ and:

$$M \leq \min_i\{c_i\} + \epsilon, \tag{2.6}$$

where ϵ is an arbitrarily small value and $0 < \epsilon$, and the values for c_i is in the closed interval $[0, 1]$. For example, a two-action automaton, is said to be ϵ-optimal if:

$$\lim_{S\to\infty} M(c_1, c_2, S) = \min(c_1, c_2), \tag{2.7}$$

where $M(c_1, c_2, S)$ is the expected penalty when we have a memory depth of S. In practical terms, it means that the automaton converges to the optimal action with a probability that is arbitrarily close to unity as the memory depth $S \to \infty$. As an example, it can be shown that Krylov automaton is ϵ-optimal in all stationary Environments, i.e., when S tends to infinity [4]. As another example, the Tsetlin automaton is only ϵ-optimal when the largest reward probability in the system is greater than 0.5, due to its equal policy treatment of penalties and rewards [48].

2.2.1 FSSA and Convergence

Convergence is an established concept in mathematics and concerns the principle of a function approaching a limit more closely as an associated parameter or variable decreases or increases. In the LA paradigm, the concept of *convergence* is an important indicator or characterization. For FSSA, we generally define the convergence as having been achieved once the automaton has reached the *innermost* state of any of the actions in its solution space. The innermost states are normally given by $iS+1$, where $i \in \{0, 1, 2, .., R-1\}$. However, the convergence criterion can be adjusted and

defined in various ways, but for FSSA schemes, it is always based on the state of the machine.

The convergence of an FSSA is related to the concept of certainty of the automaton. For example, once the automaton is in the innermost state, we can assume that it is quite certain that the current action that it is in, is the preferred action. However, the likelihood that the action to which the machine has *converged to* increases with the parameter, S. We can adjust the number of states to influence the convergence behavior and the likelihood of the automaton converging to the optimal action. The number of states in an FSSA is a trade-off between the certainty of the automaton and the efficiency of the automaton. The more states we have, the more certain the automaton will be in its final decision. However, the more states that the LA has, the more iterations it will take before the convergence is achieved.

2.3 FSSA for Partitioning Problems

In the past, solving partitioning problems included splitting a collection of numbers into subsets with the goal of reducing the difference between the subgroups' maximum and lowest sums as much as possible [60]. An example is the NP-hard, "Two-way Number Partitioning Problem", where the problem is to divide integers into two sub-sets and where the sum of the integers in all of the sub-sets should be as close to one another as possible. Using a variety of heuristic approaches in [61], the "Two-way Number Partitioning Problem" was addressed. Subsequently, the Karmarkar-Karp (KK) heuristic [62], the Complete Greedy Heuristic Algorithm [63], and the Complete Karmarkar-Karp Heuristic Algorithm [64] used a variety of strategies based on binary trees in order to resolve the issue. In recent years, the issue has been expanded to deal with vectors rather than numbers. One example of this is the "Multidimensional Two Way Number Partitioning Problem", which has been addressed by using a genetic algorithm [65] and with a metaheuristic strategy in [66].

In a more generalized manner, we can say that partitioning problems concern dividing a set of objects into groups based on some known (or unknown) criteria. Such problems have been referred to as OPPs [67, 68], as mentioned earlier in the Introduction. In this dissertation, we invoke this distinct and broader definition of *partitioning*. Although OPPs may seem similar to *clustering*, the OPPs have distinct characteristics. Clustering methods like, e.g., K-Means, usually group the objects directly based on distance metrics that are presented without any uncertainty. However, in the case of OPPs, there might be no up-front information to base the grouping on, as the information might only be obtainable over time. In addition, for an OPP, there might be no available distance metric that can be used as a foundation for the grouping. The conditions might be based on accessibility, i.e., the issue of queries of objects being accessed together. Consequently, for an OPP, we might have a stochastic underlying sequential presentation of objects accessed together, where their accessibility has an inherent meaning that they should be grouped. Another aspect is the uncertainty that often complicates the grouping in

OPPs, where the information contains stochastic or misleading leads. Consequently, the solution needs to handle the stochastics of the presented queries[3].

In what follows, we will refer to partitioning in terms of OPPs. Additionally, we divide OPPs into two different scenarios. Thus, when the problem concerns dividing objects into equally sized groups, we refer to the problem as an EPP, as already alluded to in the Introduction. For example, an EPP can concern dividing six objects into three groups, where each group has room for two objects. In contrast to EPPs, we have NEPPs, where the grouping problems concern dividing the objects into unequally-sized groups. Thus, an example of a NEPP is that of dividing six objects into two groups, where one group has room for four objects, and the other has room for two objects. In Figure 2.7, we visualize the difference between an EPP and a NEPP. The reader should also notice the terminology, where Δ^* refers to the optimal (and unknown) partitioning. Furthermore, Δ^+ refers to the partitioning found or determined by an arbitrary algorithm, and when $\Delta^+ = \Delta^*$, the solution found is said to be the optimal partitioning. The distinction between EPPs and NEPPs is essential in this dissertation, because the research is related to novel contributions for solving NEPPs, since the prior existing algorithms were only able to solve EPPs.

We define an OPP as a problem where we have O objects to be divided into K disjoint groups. The objects in the grouping problem are referred to by o_i, where $i \in \{1, 2, ..., O\}$. Further, we denote a certain group as ϱ_k, where $k \in \{1, 2, ..., K\}$. Consequently, as an example, $\varrho_1 = \{o_1, o_2, o_3\}$, means that o_1, o_2, and o_3 are all elements of partition ϱ_1. Consequently, when $\frac{O}{K}$ is an integer and there are an equal number of objects in each group, we are dealing with an EPP.

It is possible to employ LA to solve OPPs. As we proceed with the presentation of the FSSA solutions to partitioning, the reader will notice that the ways in which we might model FSSA to tackle such situations are considerably different from, for example, clustering approaches and other classic grouping techniques. As already emphasized about LA, they are able to handle stochastic Environments. Consequently, these solutions represent powerful solutions where the grouping problems are hard to determine using, e.g., merely statistical strategies.

2.3.1 Previous LA Solutions for Partitioning Problems

Two of the earliest non-LA solutions to the OPP were the Hill-Climbing method proposed in [69] and the Basic Adaptive Method (BAM) detailed in [70]. The Hill-Climbing method was based on pairing, scoring, and replacing with two different processes until no further improvements to the groupings could be obtained. Thus, the Hill-Climbing method was tedious, and it was even claimed that no real-life simulations of the method could be made in [71]. The BAM partitioned objects without having the need to specify the number of partitions. This method was

[3]We emphasize that an OPP and the criterion for objects being accessed together can be modeled and configured in numerous ways, and that even distances can be considered in this configuration as exemplified later in the dissertation.

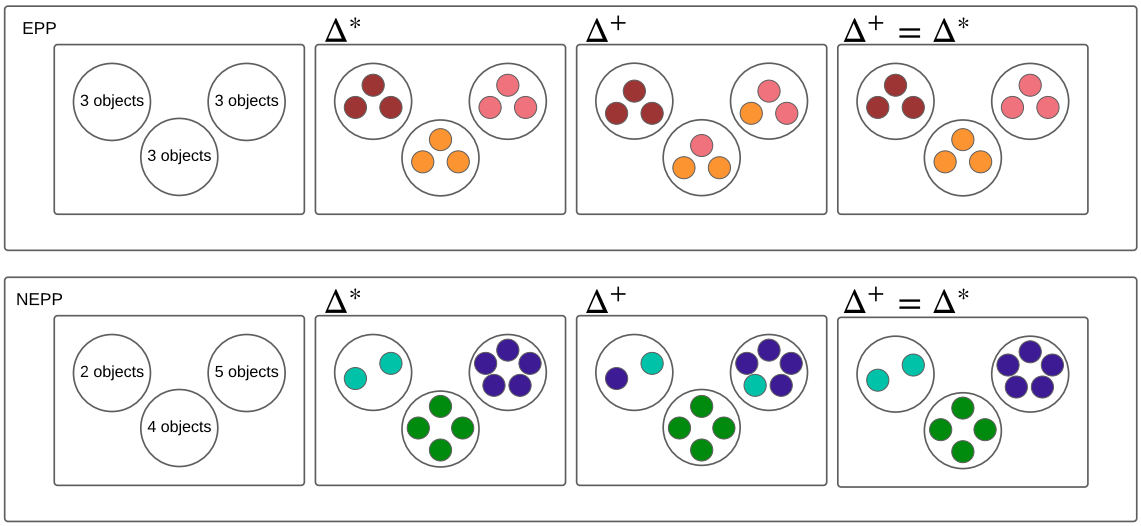


Figure 2.7: Visualization of an EPP and a NEPPn and the different terminologies used in this dissertation.

based on *queries*, where a query consisted of objects that should be grouped. In this dissertation, we denote a query by Q, where $Q = \{o_i, o_j\}$ indicates that o_i and o_j are accessed together. The BAM aimed at grouping the queried objects by bringing them closer together on a *real line*. Each object was initially assigned a real number, and their respective numbers were changed upon receiving a query. Ultimately, the partitions were determined after a certain number of queries had been considered, and this was based on the objects' nearness to one another. The BAM suffered from slow convergence, and it also required a mechanism for preventing all the objects from moving to the same group. This phenomenon negatively influenced the other objects and could block the algorithm from finding the correct or optimal partitioning.

It was later demonstrated that LA-based methods, like the Tsetlin automaton and the Krinsky automaton, could solve OPPs better than the Hill-Climbing method and the BAM, respectively. However, as demonstrated in [67], the Tsetlin and Krinsky became impractical as the number of partitions and partition sizes increased. Rather, when solving OPPs by LA, the problem needs to be modeled quite differently. Thus, in these solutions, the objects are represented as abstract objects that themselves traverse the LA's states, and the current state of such an object determines the group to which it belongs. More details about the Tsetlin and Krinsky methods are presented below.

2.3.1.1 The Tsetlin Automaton for Solving OPPs

The Tsetlin automaton was described in detail above with examples of its operation upon a Reward and Penalty as in Figure 2.3 and Figure 2.4, respectively. The reader should remember that such a machine has RS states. Thus, we have eight states in total, if we have two actions and four states per action. For a Tsetlin automaton to solve an OPP, we need to arrange the same number of actions as we

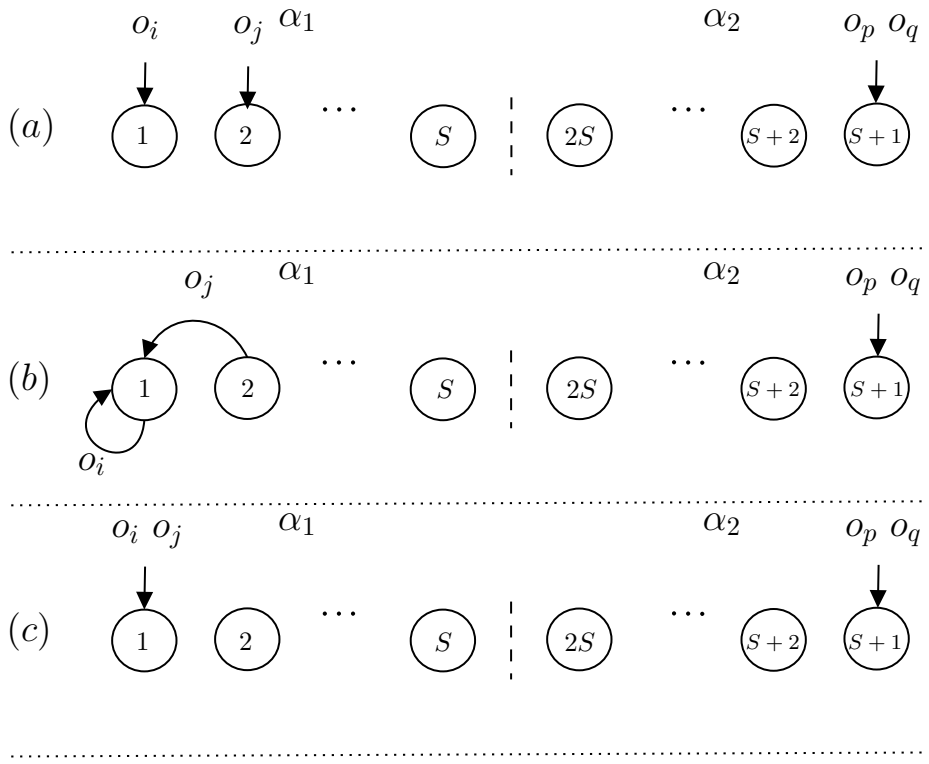


Figure 2.8: Example of the Tsetlin automaton's operation upon a Reward.

have groups in our partitioning problem, i.e., $R = K$. A Tsetlin automaton needs to be configured differently from traditional LA in order for it to solve a partitioning problem. Traditionally, the automaton itself is in a state and traverses through the states in its state space. However, to solve OPPs, the objects themselves are configured to traverse the state space of the automaton. Consequently, the objects each have a state, which determines their partition.

Let us consider an example where we have four objects that should be partitioned into two partitions. In case (a) in Figure 2.8, we can observe that o_i and o_j are in the same action. When presented with $Q = \{o_i, o_j\}$, the objects are rewarded as visualized in (b). The overall distribution of objects after the LA receiving this query, and getting a feedback from the Environment is visualized in (c). In general, for LA and partitioning problems, we use the terminology that the automaton has *converged* once all the objects are in the innermost states. The innermost states are the states furthest away from the boundary states, where an object is able to change action. In (c), we see that the machine has converged.

In Figure 2.9, we can observe the Tsetlin automaton's operation upon receiving a Penalty. In this case, the queried objects are not currently inside the same action (group). Consequently, the objects are penalized. One of the objects switches to α_1 because it is currently in the boundary state of α_2, while the other action is moved one step shallower. If we imagine the next time instant, the objects might be rewarded if we receive another query consisting of the same objects. In this way, we see that the objects that are queried manage to get together inside the same action (group) and might eventually converge.

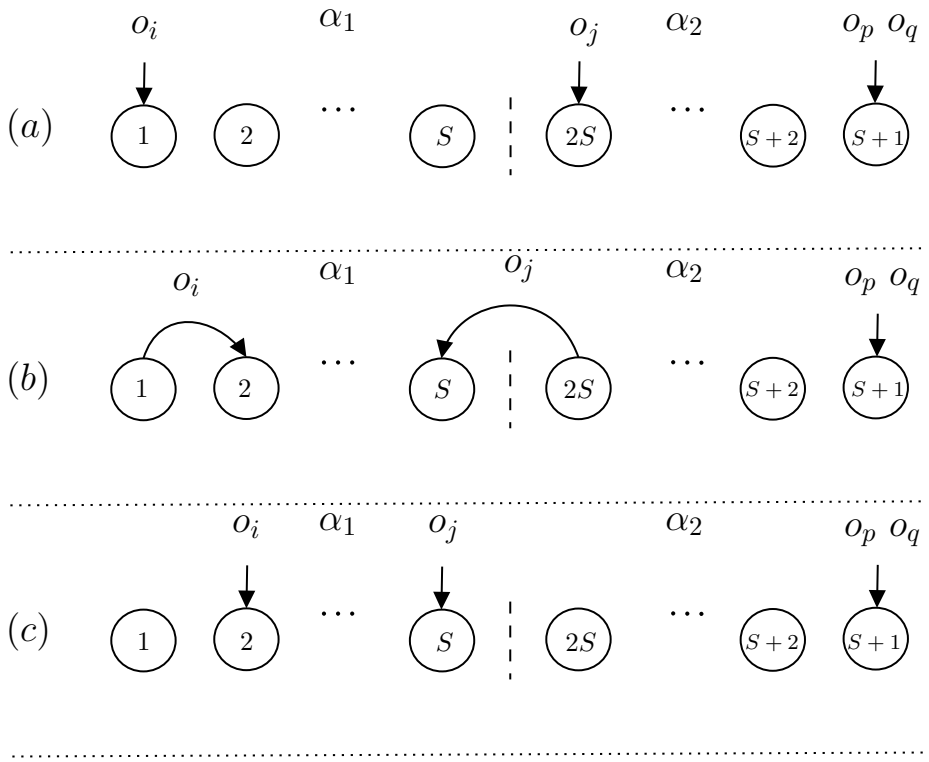


Figure 2.9: Example of the Tsetlin automaton's operation upon a Penalty.

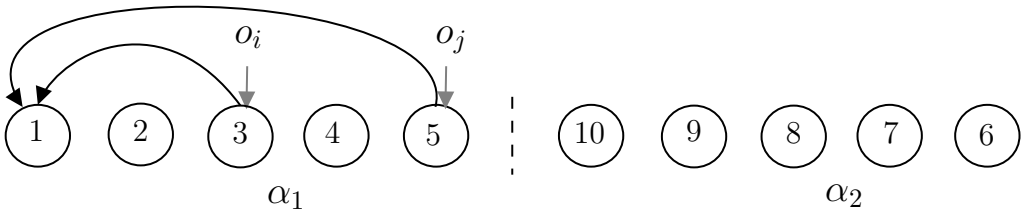


Figure 2.10: The Krinsky operation upon a Reward for an OPP.

2.3.1.2 The Krinsky Automaton for Solving OPPs

The Krinsky automaton operates similarly to the Tsetlin automaton when it comes to solving partitioning problems, as outlined above. However, the Krinsky automaton has a different transition function that is activated upon receiving a Reward. Specifically, if the queried objects are currently in one of the intermediate actions, the automaton will "jump" directly to the innermost state of the current action as visualized in Figure 2.10. This behavior allows the machine to converge faster, but the machine is also more vulnerable to obtaining an incorrect partitioning, if the presented query is noisy (as explained in more detail later), and does not represent objects that should be together.

The reader should note that there is no configuration in the Krinsky automaton (or in the Tsetlin automaton) that prevents all objects from residing and converging to the same action. This drawback prevents the optimality of these LA for solving partitioning problems, as no mechanism is available to divide the groups correctly.

Another drawback is the operation upon a Penalty, where no mechanism ensures that the queried objects are moved favorably. If two objects are queried and are in different groups, they will be penalized (because we want them to be in the same group). Thus, upon a Penalty with many groups, one object might switch to one action and the other to a completely different action. Consequently, it might require many queries before they are *fortunate enough* to switch to the same action if they are really meant to be together. If the automaton is completely deterministic in its switching between the states, they may even experience situations where they do not co-exist in the same group/action. In this case, the machine will, therefore, be prevented from converging, i.e., the objects reaching the innermost states.

2.3.2 Existing OMA Solutions for Partitioning Problems

The original OMA, further referred to as the *Vanilla* OMA, was introduced in [72] and provided a novel strategy for resolving partitioning issues using LA[4]. The OMA solutions to OPPs (specifically EPPs) are the most efficient LA solutions to such problems [38, 39]. The OMA can solve EPPs based on the concept that there are as many actions as there are groups in the grouping problem. Thus, if a grouping problem has three groups, the OMA will have three actions. In OMA, the entities to be grouped are referred to as *abstract objects*. The essential idea of OMA is that the abstract objects themselves traverse the actions in the LA, where each object's present group correlates to its current state within the action to which it belongs. The OMA assumes that, based on an underlying and unknown criterion only known to an Oracle, abstract objects queried together should be together. The rule that certain objects are queried is limited only by inventiveness, resulting in a vast array of potential applications for these algorithms.

The OMA algorithms have been applied to numerous domains. Already in the 1990s, it was used in the sphere of cryptoanalysis by Oommen and others in [30]. In this research, the OMA was employed to solve a substitution cipher by utilizing examples of the plaintext and its analogous ciphertext. In the research field of image analysis, the OMA was utilized to extract information about which images were similar in [31]. Another security-related problem was addressed in [36] and [37], respectively, where the OMA was utilized for securing statistical databases. For securing statistical databases, the OMA beat state-of-the-art algorithms for solving the Micro-Aggregation Problem (MAP).

One of the applications that is often used as an example of demonstrating the strength of the OMA algorithms is in cloud computing, and involves the problem of determining where to distribute files optimally across multiple databases or storage units based on their user's accessibility. Resolving data fragmentation in a distributed database system was solved using the OMA in [35]. The problem was NP-hard, and the database system had a graph-like structure. Another cloud

[4]The reader should note that throughout this dissertation, the term OMA refers to its paradigm itself and to algorithms, in general. The specific variants are stated when it is significant to the context.

computing-related issue was addressed in [33] and [34], where the authors utilized OMA for dividing traffic in the cloud across several machines. The OMA solution for traffic distribution achieved a 90% cost reduction compared with the existing solutions for such traffic distribution problems. Another domain where the OMA has been applied to is the domain of reputation systems. In [32], the OMA was employed to enhance the trustworthiness of reputation systems. More recently, the OMA was utilized for outlier detection in noisy sensor networks [38], and for organizing and grouping in SSLs in an adaptive manner [39].

The OMA algorithms learn in a semi-supervised manner. The algorithms are semi-supervised, because they are presented with a certain amount of information along time, but they are not shown, e.g., examples of optimal partitions (training data from which partition information about the groups for the objects can be gleaned). More specifically, the OMA algorithms require input information in terms of *queries*, similar to the BAM. Each query consists of pairs of objects[5], and the assumption is that these have an underlying inherent property that means they should be grouped. Similar to the Tsetlin and Krinsky automata, the objects themselves traverse the states within the OMA, and if the queried objects are currently inside the same action, they are rewarded, and penalized if they reside in different actions. In this way, the OMA are able to group the objects that are mostly queried together [38], and attain to a hopefully-optimal grouping.

One of the OMA's strengths is its ability to operate in stochastic Environments. Thus, even if it receives misleading queries, it can attain an optimal partitioning with a very high probability [38]. Misleading queries are queries of objects that *do not belong together* and are referred to as *noisy queries* (or *noise*) within the OMA paradigm. In simulations, we can mimic stochastic Environments by adjusting the level of noisy queries presented as input to the automaton. We emphasize that, for a real Environment, we have no way of knowing whether a query is noisy or not. As an example, if o_1 and o_2 are in one of the groups in Δ^*, and o_3 and o_4 in another, the query $Q = \{o_2, o_3\}$ is a noisy query. On the other hand, $Q = \{o_1, o_2\}$ is not a noisy query.

In the OMA paradigm, we have different variants with distinct characteristics. Thus, we have the Vanilla OMA, the Enhanced OMA (EOMA), the PEOMA[6], the Transitivity PEOMA (TPEOMA). As briefly mentioned above, the Vanilla OMA was first proposed by Oommen *et al* in [72]. The Vanilla OMA variant had an issue that was resolved in the improved EOMA variant [73]. Consequently, the EOMA variant was continued, and the improved PEOMA variant was proposed in [17] and [18]. The PEOMA utilizes the Pursuit concept, where the probabilities about likely query pairs are estimated and utilized in order to expedite the OMA's convergence process. The latest variant, the TPEOMA, introduced in [74], utilizes

[5]The problem of "queries of increased size" is considered outside the scope of this work, and is rather considered as a problem for further research.

[6]We also have the POMA version, but this version has the same impediment as the OMA explained subsequently, and therefore, the PEOMA variant is preferred over the POMA. A detailed explanation of the POMA can be found in [16].

the transitivity concept. The latter concept makes the algorithm perform even better in certain Environments in terms of the required number of queries before attaining convergence [38].

Although all of the applications listed above demonstrate the OMA's versatile applicability to real-life issues, their impediment of only being able to handle equally sized groups, might have limited their overall applicability. As demonstrated by the research contributions presented later in this dissertation, their ability to handle even more complex group structures can greatly enhance the usefulness of OMA methods in a variety of settings. Two approaches that have been proposed to address this limitation is to use a modified versions of the OMA method that allows for the inclusion of group sizes that are not equal. These research contributions will be addressed later. However, it is important for the reader to first understand the concept motivating the existing OMA algorithms. Therefore, in the subsequent sections, we will briefly present the different existing types of OMA.

2.3.2.1 *Vanilla* OMA

Similar to the Tsetlin and Krinsky methods explained earlier, the Vanilla OMA needs as many actions as there are groups in the partitioning problem, and has S states per action. Likewise, the objects themselves traverse the states, and their state determines their respective group. However, as highlighted, the Tsetlin and Krinsky methods had drawbacks and needed more efficiency. Therefore, a tailored LA solution for partitioning was needed. Such a solution was introduced in [67, 72], namely the Vanilla OMA. The operation of the Vanilla OMA is based on the concept that the LA is presented with queries and that each of the queried objects can stay in a state, move from one state to another, or migrate to another action. The transition function is dependent on whether the LA receives a Reward or a Penalty and includes changes to the locations of the queried objects. Unlike the Tsetlin and Krinsky method, the Vanilla OMA includes policies for objects other than the ones that are queried.

In Figure 2.11, we visualize the concept of the OMA. As we can observe, we have a system that generates queries and inputs them to the LA. The OMA then reports its current object distribution (the states or actions of the queried objects) and the query it received from the system. Consequently, the Environment will respond with a Reward if the queried objects are currently in the same action (group) of the OMA. If the queried objects are not in the same action (group), the Environment will respond with a Penalty. Based on the feedback from the Environment, the queried objects are rewarded or penalized. Whether an object changes its state or not, depends primarily on whether it currently is in one of the boundary states or in one of the innermost states. More specifically, $\phi = (k-1)S + 1$ indicates the innermost states and $\phi = kS$ indicates the boundary states, where $k \in \{1, 2, ..., K\}$ and k denotes the partition ϱ_k.

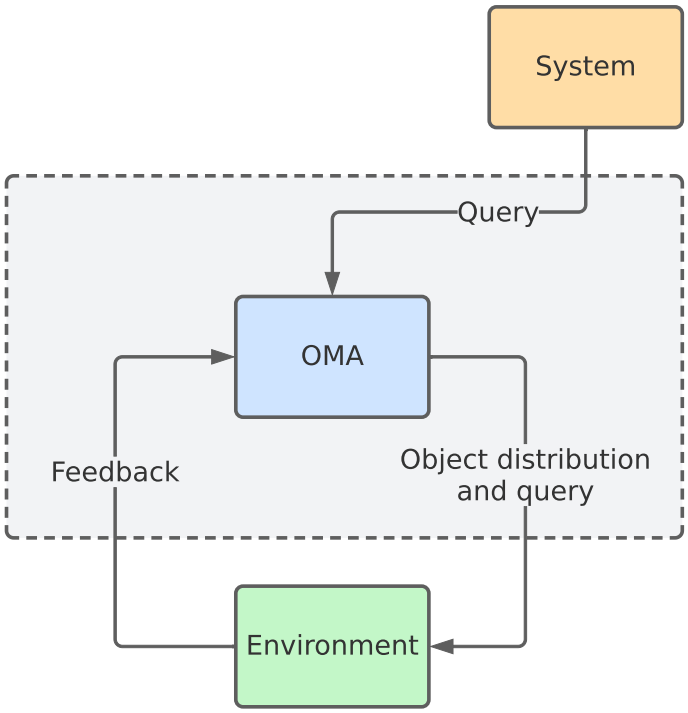


Figure 2.11: Schematic of the overall OMA concept.

An overall description of the Vanilla OMA[7] is presented in Algorithm 1 [75]. We emphasize that the Vanilla OMA (and its other existing variants) can only solve EPPs. Thus, it can solve groups of size $\frac{O}{K}$, where the size is an integer, and where we have an equal number of objects in each group. One of the reasons for this limitation is that we want the objects to be separated into distinct groups and not allow for the possibility of one group residing in the same action as another group. Without the relation that the actions (groups) need to have the same number of objects, the swapping operation of the penalties becomes void. Additionally, we only have information on objects that are accessed together, meaning that it is hard to know which objects are to be moved without the equi-principle being followed.

The Vanilla OMA is initialized by distributing the objects randomly among all the states of the automaton, with $\frac{O}{K}$ objects in each action. Then, the automaton considers the queries that are submitted as inputs according to its policy, until convergence is achieved or a maximum number of queries has been considered. The machine is said to have *converged* once all the objects are in the innermost states of the machine, and the automaton reports the partitioning that it has discovered based on the objects' states. Checking the current partition of an object can be accomplished mathematically by dividing the object's current state by the number of states per action, evaluating the "Floor" function of the resulting value, and adding unity, i.e., $\lfloor \frac{\phi_i - 1}{S} \rfloor + 1$. The most important functionality of the OMA is

[7]The reader should note that only the details of the Vanilla OMA algorithm are presented here, and that the others (EOMA, PEOMA, etc.) are omitted (their algorithmic descriptions) to avoid repetition, and because they are part of the papers presented in later parts of this dissertation, and because some of them receive less focus in the dissertation. For the interested reader, the algorithm descriptions can be found, in detail, in [75].

Algorithm 1 The Vanilla OMA Algorithm

Input:

- The set of objects, and an object is denoted as o_i with $i \in \{1, 2, ..., O\}$.
- S states per action (group).
- A sequence of queries, where one query is denoted as $Q = \{o_i, o_j\}$.

Output:

- A partitioning ($\mathcal{K} = \Delta^+$) of the O objects into K partitions, and $\mathcal{K}$ is the set of partitions, $\mathcal{K} = \varrho_1, \varrho_2, ..., \varrho_K$, and, e.g., $\varrho_2 = \{o_2, o_5\}$.
- ϕ_i is the state of o_i. It is an integer in the range $\{1, 2, ..., KS\}$.
- If $(k-1)S + 1 \leq \phi_i \leq kS$ then o_i is assigned to ϱ_k, which is done for all $i \in \{1, 2, ..., O\}$ and $k \in \{1, 2, ..., K\}$ [38].

Initialize ϕ_i, $\forall\ i,\ i \in \{1, 2, ..., O\}$
while not converged **do**
  Read query $Q(n) = \{o_i, o_j\}$
  if $\lfloor \frac{\phi_i - 1}{S} \rfloor + 1 = \lfloor \frac{\phi_j - 1}{S} \rfloor + 1$ **then** ▷ The objects are rewarded
    Vanilla OMA Process Reward($\{\phi_i, \phi_j\}, Q(n)$)
  else ▷ The objects are penalized
    Vanilla OMA Process Penalty($\{\phi_i, \phi_j\}, Q(n)$)
  end if
end while
Output the final partitioning based on ϕ_i, $\forall\ i$.

how it processes Reward and Penalty inputs from the Environment, and a basic description of these phenomena can be summarized as follows:

- **Reward** - the queried objects are in the same action (group)
  - Case 1: None of the objects in the query are in an innermost state. In this case we move both objects one step towards the innermost state.
  - Case 2: One of the queried objects is in an innermost state. In this case, we let the object in the innermost state remain in its state and move the other queried object one step towards the innermost state of its action.
  - Case 3: Both queried objects are in the innermost state. Here we permit both the objects to remain in their current states.

  We describe the Reward operation of the Vanilla OMA in Algorithm 2 [75].

- **Penalty** - the queried objects are **not** in the same action (group)
  - Case 1: None of the queried objects are in their boundary states. In such a case, we move both the objects one step towards the boundary states in their respective actions.
  - Case 2: One of the queried objects is in a boundary state, and the other is not in a boundary state. In this case, we move the non-boundary object

one step towards the boundary state of its action, and let the object at a boundary state remain in its current state.

– Case 3: Both queried objects are in boundary states. The new position of the objects is determined as follows. Let (by uniform sampling) one of the objects be the *staying* object and the other be the *moving* object. Then, let the *moving* object switch to the state of the staying object, and take a non-queried object (the one closest to the boundary - and if no single object closest to the boundary can be determined, we choose one by uniform sampling) from the *staying* object's action and move it to the state that the *moving* object came from.

We describe the penalty operation of the Vanilla OMA in Algorithm 3 [75]. We emphasize that in the Vanilla OMA version, both queried objects need to be in their respective boundary states for them (one of them in two-object query configurations) to switch its action.

Algorithm 2 Vanilla OMA Process Reward($\{\phi_i, \phi_j\}, Q$)

Input:

- The query pair $Q = \{o_i, o_j\}$.
- The states of the objects in Q ($\{\phi_i, \phi_j\}$).

Output:

- The next states of o_i and o_j.

if $\phi_i \bmod S \neq 1$ **then** ▷ Move o_i towards the innermost state
    $\phi_i = \phi_i - 1$
end if
if $\phi_j \bmod S \neq 1$ **then** ▷ Move o_j towards the innermost state
    $\phi_j = \phi_j - 1$
end if

2.3.2.2 Enhanced OMA (EOMA)

One issue with the Vanilla OMA is that it suffers from the *Deadlock Situation*[8], where the automaton can become stuck in an endless loop due to which it is prevented from converging in noise-free Environments. Additionally, the Vanilla OMA suffers from slow convergence. Therefore, the authors in [73] presented the EOMA that does not have the same impediment. The significant differences between the EOMA and the Vanilla OMA are that the objects are initially distributed differently, the convergence criterion is adjusted, and the automaton's operation upon a Penalty feedback is adjusted. In this way, the EOMA represents a more robust method than

[8]For the interested reader, a detailed explanation and an example of the Deadlock Situation can be found in [75].

Algorithm 3 Vanilla OMA Process Penalty($\{\phi_i, \phi_j\}, Q$)

Input:

- The query pair $Q = \{o_i, o_j\}$.
- The states of the objects in Q ($\{\phi_i, \phi_j\}$).

Output:

- The next states of o_i and o_j.

if ϕ_i mod $S \neq 0$ and ϕ_j mod $S \neq 0$ **then** ▷ Neither are in boundary
    $\phi_i = \phi_i + 1$
    $\phi_j = \phi_j + 1$
else if ϕ_i mod $S \neq 0$ and ϕ_j mod $S = 0$ **then** ▷ o_j is in boundary
    $\phi_i = \phi_i + 1$
else if ϕ_i mod $S = 0$ and ϕ_j mod $S \neq 0$ **then** ▷ o_i is in boundary
    $\phi_j = \phi_j + 1$
else ▷ Both are in boundary states
    $temp = \phi_i$ or ϕ_j ▷ Store the state of *moving* object, o_i or o_j
    $\phi_i = \phi_j$ or $\phi_j = \phi_i$ ▷ Put *moving* object and *staying* object together
    o_l = *unaccessed* object in group of *staying* object closest to boundary
    $\phi_l = temp$ ▷ Move o_l to the old state of *moving* object
end if

the Vanilla OMA. Therefore, it has been used as a foundation for other extensions, and also the enhancements of this Ph. D. study.

One can summarize the overall new functionality of the EOMA, when compared with the OMA, as follows:

- **Different Initialization of Objects:** In the EOMA, the objects are initialized into the boundary states of the automaton. Thus, $\frac{O}{K}$ objects are distributed randomly (with a uniform probability distribution) into each of the automaton's boundary states. Consequently, instead of distributing the objects randomly into all of the KS states, the EOMA distributes them among the K boundary states of the automaton. In this way, the objects can easily change the action that they initially reside in, upon receiving the first queries, reducing the average number of queries required before convergence in an overall sense.

- **Different Penalty Operation:** The EOMA possesses a different strategy for handling of the objects presented in a query, upon a certain object distribution for a Penalty. We visualize this distinct Penalty operation in Figure 2.12, where we are presented with a query, and one of the objects queried is in the boundary state, and the other is in a non-boundary state. Consequently, the EOMA operation lets us move the boundary object to the boundary state of the action of the non-boundary object in the query. In this way, the objects are able to switch actions without both objects in the query needing to be in their boundary states. This allows the EOMA to be much more flexible and efficient than the Vanilla OMA. Let us consider the case that we have

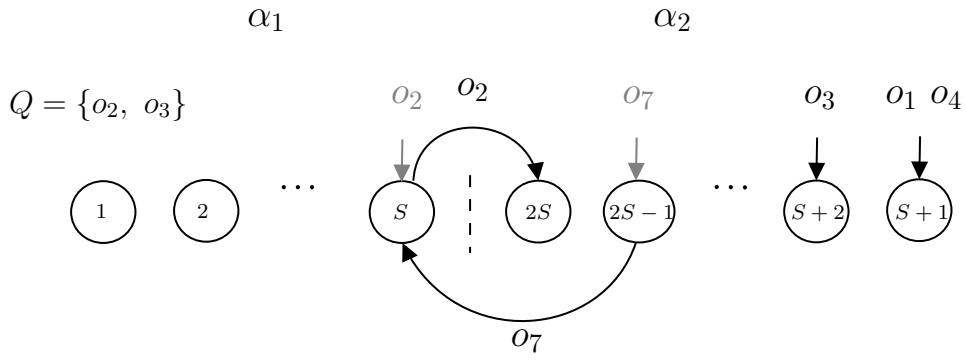


Figure 2.12: The Penalty operation of the EOMA.

o_1, o_3 and o_4 inside one action (as in the example presented in Figure 2.12) with the Vanilla OMA in mind. In Δ^*, o_2 should also be together with these objects. However, o_2 is currently in another action (group). The automaton will probably be presented with queries consisting of objects that are already correctly grouped, pushing, e.g., o_3 towards the innermost state. On the other side, queries consisting of o_2 together with any of the objects that are already grouped will (incorrectly) push them towards the boundary. For the Vanilla OMA, o_2 will only be able to move to the *correct* action once either o_1, o_3 or o_4 is moved to a boundary state, and a query consisting of one of these and o_2 is presented to the automaton. Additionally, e.g., o_3 might even be randomly switched to the action of o_2, resulting in a lot of iterations back and forth before the objects are correctly grouped. Therefore, the EOMA operation is much more effective, because it would directly move o_2 to the action of o_1, o_3 and o_4.

- **Different Convergence Criterion:** In the Vanilla OMA, all objects need to reside in the innermost states for the automaton to be seen to have *converged*. The convergence criterion for the EOMA is extended to include the second innermost states. Thus, the convergence criterion for the EOMA is that all objects reside in the innermost or the second innermost state in the actions of the automaton. Consequently, the EOMA has a more relaxed criterion for convergence when compared with the Vanilla OMA. This distinct criterion for convergence makes it easier for the algorithm to converge, especially in noisy Environments.

In the paragraphs above, we have briefly explained the phenomena that makes the EOMA different from the Vanilla OMA. The rest of the EOMA's concept and operation are similar to the Vanilla OMA. Thus, the EOMA has the same Reward function as the Vanilla OMA (as described in Algorithm 2), but the overall operation and the Penalty functionality are different.

Another detail about the Penalty operation of the EOMA that we would like to emphasize, is that it has been done in the Literature in two different ways. In Figure 2.12, the boundary object is moved to the boundary state of the non-boundary object, similar to the operation explained in [38]. However, as explained

in [39], upon such an object distribution as exemplified above, the boundary object can also be moved directly to the same state as the non-boundary object or switch state directly with the non-queried object that is moved to the boundary object's action. These differences have a minor conceptual impact on the overall operation of the EOMA but might result in a different average number of iterations before convergence.

2.3.2.3 Pursuit EOMA (PEOMA)

Noisy queries are a complicating factor in stochastic Environments, and the algorithms in the OMA paradigm can handle such stochastic behavioral patterns. However, the PEOMA algorithm is even more robust against noisy queries because it utilizes the *Pursuit concept*. In LA, the Pursuit concept concerns pursuing the most likely "Best" action [50]. The PEOMA is an extension of the EOMA and in the former, the Pursuit concept is implemented a bit differently when compared with the traditional LA which posses the same concept. In the case of PEOMA, we filter out noisy queries using estimates, and only present queries to the automaton that are likely to be "Correct" [16, 17, 18, 38]. In this way, the PEOMA represents a more effective solution, especially when high levels of noise are encountered [38], because fewer misleading queries are presented to the automaton.

The way that the Pursuit concept is implemented in the PEOMA is through a frequency ("Pursuit") matrix that is updated and checked in the process of learning. This matrix, denoted as $\mathcal{M}$, has O rows and O columns for a system that presents queries consisting of two objects. In more detail, the matrix maintains the probability of each pair of objects being presented to the automaton from the system based on previous queries, and in this way, we can determine the likelihood of the queries of being *noisy*. If a query has been presented less often, it is more likely to be a noisy query. Thus, if the $Q = \{o_1, o_2\}$ has been presented 100 times, and the automaton has considered 1,000 queries in total, the likelihood (or sampled probability) of such a query being presented is 0.1. Likewise, if 10 out of the 1,000 queries is $Q = \{o_1, o_3\}$, the sampled probability of that query being presented is 0.01, meaning that this query is less likely to be presented from the system when compared with $Q = \{o_1, o_2\}$. These probabilities are tracked during the operation of the PEOMA. In Figure 2.13, we visualize, using "dummy" entries, the concept of the frequency matrix, $\mathcal{M}$.

$$\mathcal{M} = \begin{array}{c} \\ o_1 \\ o_2 \\ \vdots \\ o_O \end{array} \begin{array}{c} \begin{array}{cccc} o_1 & o_2 & \cdots & o_O \end{array} \\ \begin{bmatrix} 0 & 100 & \cdots & 10 \\ 100 & 0 & \cdots & 20 \\ \vdots & \vdots & \vdots & \vdots \\ 10 & 20 & \cdots & 0 \end{bmatrix} \end{array}$$

Figure 2.13: The frequency matrix for query occurrences in the PEOMA.

The reader should note that $\mathcal{M}$ is symmetric along the diagonal and that the diagonal itself has no meaning, since o_i together with o_i has no purpose in a grouping problem, implying that the diagonal consists of zeros. The symmetry implies that we have two entries for each object pair (query). Consequently, we update both entries as we are presented with a query from the system. In reality, only half of the matrix (either side of the diagonal) is actually needed, because we do not consider the order of the objects in the query in the OMA processes. The frequency matrix is checked before the query is considered by the automaton and is sent to the Environment. If the number of times a query has been presented divided by the total number of queries that the system has generated is below a certain threshold we do not consider the query, and thus, this query is *filtered out*. We emphasize that, when implementing the PEOMA, it is important to make sure that the query probabilities are calculated correctly[9].

The PEOMA has two phases. Initially, some queries have to be considered/sampled before we start using $\mathcal{M}$ for the task of filtering. This initial phase is called the *Estimation phase*. The number of queries that should be considered in the Estimation phase, denoted by κ, can be adjusted, and it is a parameter whose value can be set through the following formula [38]:

$$\kappa = \left(\left(\frac{O}{K} \right)^2 - \left(\frac{O}{K} \right) \right) K. \tag{2.8}$$

In the Estimation phase, the PEOMA acts as a normal EOMA, but updates $\mathcal{M}$ based on the incoming queries. After the Estimation phase, we enter the *Filtering phase* (also called the *Thresholding phase*), where the likelihood of the queries are considered before they are processed. We denote the threshold for whether a query should be processed or not as τ, where $\tau < \frac{1}{O}$, and is an arbitrarily small value. The threshold for the filtering needs to be adjusted according to the Environment and the value can be adjusted according to the probabilities after κ queries has been considered. Tuning κ and τ can be difficult, because it is a trade-off between efficiency and accuracy. If we use a large κ, the probabilities become more certain, but this requires more iterations. Likewise, with τ, a large value might yield a scenario where only high quality queries are processed by the automaton, but at the same time many queries that are actually useful might be filtered out, reducing the overall system efficiency.

2.3.2.4 Transitivity Pursuit EOMA (TPEOMA)

As an extension to the PEOMA, the Literature reports the most recent machine in the OMA paradigm, namely the TPEOMA [74]. In this OMA version, we utilize the concept of *Transitivity*. By considering relations among the objects which can be deduced from $\mathcal{M}$, we can *generate* and present artificial information to the automaton even if the system presenting the automaton with queries is dormant.

[9] The reader should remember that, the sum of all the elements in $\mathcal{M}$ represents the number of queries times two, due to the symmetry.

In this way, the TPEOMA is a powerful solution when we have sparse information from the query-generating system. In more detail, the TPEOMA is based on the principle that if we know that, e.g., o_1 and o_2 should be together in a group, and that o_2 and o_3 should be together in a group, then we can deduce that o_1 and o_3 should also be together.

The interesting feature of the TPEOMA is that we can obtain the transitivity relations from the frequency matrix, $\mathcal{M}$. To utilize the transitivity concept, we introduce a new parameter for handling these relations, which is a threshold for considering a transitivity relation. Let us denote the threshold parameter for the transitivity relations as τ_t. By way of example, if the system generates a query consisting of o_1 and o_2, we will first check the frequency matrix and see whether the probability of that query being presented is above τ. If the query likeliness is above τ, we let the automaton process that query. Additionally, we will consider all combinations of o_1 and o_2 together with other objects in the matrix. If there exist relations with either o_1 or o_2 and other objects that have a probability above τ_t in the matrix, we will input these as artificial queries to the automaton. As an example, if the system consists of o_3 or o_4 in addition to o_1 and o_2, we will consider the combinations: $\{o_1, o_3\}$, $\{o_1, o_4\}$, $\{o_2, o_3\}$, $\{o_2, o_4\}$. If any of these combinations result in a probability of being accessed together above τ_t, we will present them also to the automaton for processing.

Normally, we configure τ_t to be stricter than τ. However, we should always adjust the threshold parameters according to the grouping problem, and for an educated guess for the transitivity threshold parameter, we can use the following equation [74]:

$$\frac{1}{O(O-1)} < \tau_t < \frac{K}{O(O-K)}. \tag{2.9}$$

The TPEOMA is identical to the PEOMA, but when we consider whether the probability of a query is above a threshold, and it is, we also check the queried objects' transitivity relations. If the transitivity relations have a likelihood of being accessed above the τ_t threshold, we additionally generate these objects as artificial queries for the automaton to process.

The TPEOMA represents the state-of-the-art algorithm in the OMA paradigm, and the authors in [39, 74] demonstrated that the TPEOMA required up to two times less number of "real" queries before convergence was achieved, when compared with the Vanilla OMA[10].

2.4 Chapter Summary

In this chapter, we have explained the concept of LA and, specifically, the sub-group of FSSA schemes. Consequently, we have outlined the characteristics of FSSA, with

[10]The reader should note that tuning the thresholds and κ parameter can be a complicated and time-consuming task. There is currently no optimization setup to determine these values exactly, and this is considered to be outside the scope of this dissertation.

examples of their operation in random Environments. As emphasized in the chapter, the FSSA schemes have a finite state space, a finite number of actions, and a finite set of possible feedback from the Environment. FSSA are independent of time in terms of their transition and output functions and are so-called state-output automatons, where the machine's state determines its action. In addition, we have discussed partitioning problems and outlined different methods for solving them using LA. In particular, we have presented the Vanilla OMA and its subsequent enhancements. The OMA algorithms are essential to this dissertation, as improvements to algorithms within the OMA paradigm are some of the novel and pioneering contributions of the work and publications within this Ph. D. study.

Chapter 3

Variable Structure Stochastic Automata (VSSA)

LA mimic the mechanisms of *learning*. In order to learn a preferred behavior, the learning unit selects actions and interacts with its teacher, namely, the Environment. As alluded to earlier, for the FSSA type of LA, the *memory* of the automaton is maintained in states, and the machine's current state determines its behavior. In this chapter, we will discuss the VSSA type of LA.

The VSSA type of LA is fundamentally different from the FSSA type because the learning unit's memory and behavior are maintained and determined by a *probability vector* (the so-called action selection probability vector) [4]. This probability vector keeps track of the different actions and their likelihood of being selected by the automaton. The machine's behavior will change by interacting with the Environment, and eventually, the automaton will, hopefully, *learn* the optimal action from the set of actions within its operating sphere, with an arbitrarily high probability. More specifically, the action selection probability vector determines the machine's chosen actions, and the vector is consequently updated based on the feedback from the Environment for the automaton to *learn* the preferred behavior.

An FSSA has to go through a number of distinct states before it can explore another action. In contrast, a VSSA can explore new actions in each iteration according to its action selection probability vector, which speeds up the process of exploring the action probability space. Within the paradigm of VSSA, there are different kinds of algorithms, and the VSSA schemes can be analyzed mathematically by representing their Markovian behavior as being either ergodic or absorbing. Further, the machine can operate in a continuous or discrete manner, and the updating function of the action selection probabilities, which are based on the Environment's feedback, can be carried out in a linear or non-linear manner [4, 8, 7]. Subsequent enhancements to the VSSA types include incorporating estimates of the action probabilities and utilizing the *Pursuit* concept. In addition, another innovation consists of incorporating structure into the machines' operation. In what follows, we will elaborate on these concepts and methodologies. Specifically, we will discuss the idea of convergence in relation to VSSA, as well as the methods by which we may assess the performance of these schemes in a variety of settings and configurations.

3.1 The Concept of VSSA

The VSSA machines are characterized by their ability to keep track of the interactions with the Environment through a probability vector that is maintained to encapsulate its memory[1]. The action chosen by the automaton is determined by means of uniform random sampling of this probability vector, making these algorithms more versatile than the FSSA schemes in terms of exploration. For VSSA schemes, the action probabilities are updated at every instant based on every interaction with the Environment [4]. For example, the probability of an inferior action could be decreased upon receiving a Penalty from the Environment. Likewise, upon a Reward, the likelihood of that action being selected could be increased. In this way, the machine *learns*. In general, the VSSA schemes maintain a single probability for each of the actions in the machine's sphere of operation. These types of LA suffer from a slow convergence phenomenon when the *action space* becomes large, and this impediment is, discussed, in more detail, later.

In formal terms, we are able to express a VSSA by using the four variables in the following way [4]:

$$\Theta = \{\mathcal{A}, \mathcal{B}, P(n), \mathcal{T}\}, \tag{3.1}$$

where we recognize from the earlier chapter that $\mathcal{A}$ and $\mathcal{B}$ are the set of actions and the set of possible feedbacks from the Environment, respectively. The action probability vector is denoted as $P(n)$, and we refer to its components as $p_i(n)$, where $i \in \{1, 2, ..., R\}$, and $p_i(n)$ is the probability of the LA selecting action i at time n. Furthermore, $\mathcal{T}$ is the machine's updating algorithm. The updating algorithm for VSSA is different from the one utilized in the case of FSSA. In VSSA, the machine operates by means of the probability vector, while FSSAs are defined by their states and the corresponding state transition and output mappings. We can define the updating algorithm, i.e., the reinforcement scheme in VSSA, as follows:

$$P(n+1) = \mathcal{T}[P(n), \mathcal{A}(n), \mathcal{B}(n)], \tag{3.2}$$

where the action probabilities at the next time instant, $n+1$, are updated based on the current action probabilities, $(P(n))$, the action that was selected as input to the Environment at time n ,$(\mathcal{A}(n))$, and the response from the Environment based on that input $(\mathcal{B}(n))$. Function $\mathcal{T}$ indicates a mapping of the updating functionality.

Reinforcement schemes can be classified based on whether they are expedient, ϵ-optimal or optimal, as established in Section 2.2. Another classification of VSSA is in regards to the nature of $\mathcal{T}$, i.e., the updating function. Furthermore, VSSA schemes cannot be mathematically represented and analyzed using Markov chains since the state space is not finite any longer. The VSSA classifications mentioned

[1]The reader should note that we present here only a brief overview of the VSSA paradigm. This is in the interest of brevity, and in order to reduce the repetition of information included in the subsequent papers presented in this dissertation. We particularly emphasize that all the concepts presented here are discussed within an informal framework so that the concepts can be easily understood. The formal mathematical details are thus omitted, but included in the accompanying papers published in the Literature.

here are based on the theory expounded in [4], and the interested reader can find more information there.

As an example, when $\mathcal{T}$ in Equation (3.2) is a linear function, the updating algorithm is said to be linear, and the automaton itself is referred to as being *linear*. More specifically, if the multiplication by a constant (that is less than unity) is used to adjust the action probabilities, the probability is adjusted in a linear manner. However, if $\mathcal{T}$ is a non-linear function, e.g., if any action probabilities are adjusted in quadratic or polynomial manner, we say that the VSSA scheme is non-linear.

We emphasize that the invention of the VSSA provided the field with a quantum increase in the speed of LA [2]. One of the reasons why these algorithms are notably faster when compared to the FSSA variants is due to the enhanced stochastic exploration of the action space. Unlike the FSSA schemes, that need to move through the states in order to explore new actions and which can only *switch* to another action when it is in a boundary state, VSSA can explore new actions at *every* time instant according to their action probabilities. In addition, the flexibility of choosing various updating functions in the case of VSSA allows for an *unlimited* number of possible updating configurations. To bring the thread back to grouping problems, VSSA can also be used for this purpose, but it is far inferior compared to the FSSA because it almost immediately reaches its limit on the number of objects that it can manage.

3.1.1 Continuous Algorithms

We have different types of VSSA schemes. The first type of VSSA that was introduced in the Literature was the continuous type [4]. The continuous type of VSSA has a continuous updating of the action probabilities. More specifically, the *learning*, i.e., the updating of the action probabilities, are done in such a way that the probabilities themselves can attain *any* value within the interval $[0, 1]$. A continuous updating rule is a function where the likelihood of the inferior actions upon a Reward are reduced, for example, in the linear case, by multiplying them with a parameter, often referred to as the *learning parameter*, and the rewarded action's likelihood are the resulting difference from unity and the sum of the probabilities of the other actions. Let us further denote the learning parameter as λ.

Two prominent VSSA schemes are the Linear Reward-Penalty (L_{R-P}) and Linear Reward-Inaction (L_{R-I}) schemes. The first variant has a linear updating scheme for the probabilities and considers both penalties and rewards. The latter is also a linear scheme, but it only processes the rewards and ignores the penalties. Both of these variants are simple schemes. However, the L_{R-I} has demonstrated better performance in terms of the average number of iterations required before the machine has converged [2] when compared to the L_{R-P} scheme. Therefore, the L_{R-I} version is often preferred over the L_{R-P} scheme. In the Reward-Penalty scheme, we can adjust the *learning parameters* and treat penalties and rewards differently. Thus, we can have two different values for λ, where one is used upon a Reward (e.g., λ_1), and the other is used upon a Penalty (e.g., λ_2). However, only when rewards and

penalties are treated equally, i.e., $\lambda_1 = \lambda_2$, can its properties be easily analyzed.

Let us consider the L_{R-P} scheme in more detail, where we consider the case where we have two actions in the machine's solution space. We represent the probabilities of the machine choosing the different actions by $P(n) = [p_1, p_2]^T$. Let λ_1 be an arbitrarily small non-zero value and $\lambda_1 \in (0, 1)$. Furthermore, we define $\lambda_2 = \lambda_1$. Because the learning parameters are identical in the case of a symmetric L_{R-P} scheme, it is not really necessary to operate with two learning parameters. However, in order to clearly explain the concept, in more detail, we utilize both notations in the details below. The L_{R-P} scheme follows the LA principle as depicted in Equation (3.1), and it will choose an action by sampling randomly based on $P(n)$. Once an action is chosen as the output, it is provided to the Environment, and the Environment will respond with a stochastic feedback. Let us assume that the feedback is from a binary set, where the Environment can respond with a Reward or a Penalty. Based on the feedback from the Environment, the two-actions L_{R-P} will update its behavior, and this reinforcement updating scheme can be specified as follows [4]:

$$p_j(n+1) = (1 - \lambda_1)p_j(n) \text{ for } j \neq \text{i},$$
$$p_i(n+1) = 1 - p_j(n+1),$$

for the case that α_i is chosen and receives a Reward ($\beta_1 = 0$) from the Environment. On the contrary, upon α_i being selected and receiving a Penalty, we have:

$$p_i(n+1) = (1 - \lambda_2)p_i(n),$$
$$p_j(n+1) = 1 - p_i(n+1) \text{ for } j \neq \text{i}.$$

Thus, when α_i is selected, and it is penalized, the probability of that action being selected in the next time instant is decreased, and vice versa.

The continuous type of LA suffers from slow/sluggish convergence[2], because, as any of the elements in the probability vector approaches unity, the change in the probability vector with every update becomes correspondingly smaller. Let us consider the case of an L_{R-I} automaton with two actions. For the sake of simplicity, assume that $P(0) = [0.5, 0.5]^T$, that only α_1 is selected, and that it is rewarded in every iteration of the algorithm. Furthermore, for this purpose of demonstration, let us define $\lambda = 0.01$. In Figure 3.1, we visualize the probability development for α_1. As we can observe from the graph, the probability increases less rapidly as we approach unity on the y-axis. We demonstrate the same principle in Figure 3.2. However, in the latter figure, we demonstrate the *change* in the probability itself between the iterations. Again, we clearly see that the change in probability becomes less as the probability approaches unity.

3.1.2 Discretized Algorithms

As explained above, the first quantum improvement in speed and efficiency in LA was achieved when VSSA came into play. The first variant of VSSA was the continuous

[2]The details of convergence for VSSA algorithms are explained in more detail below.

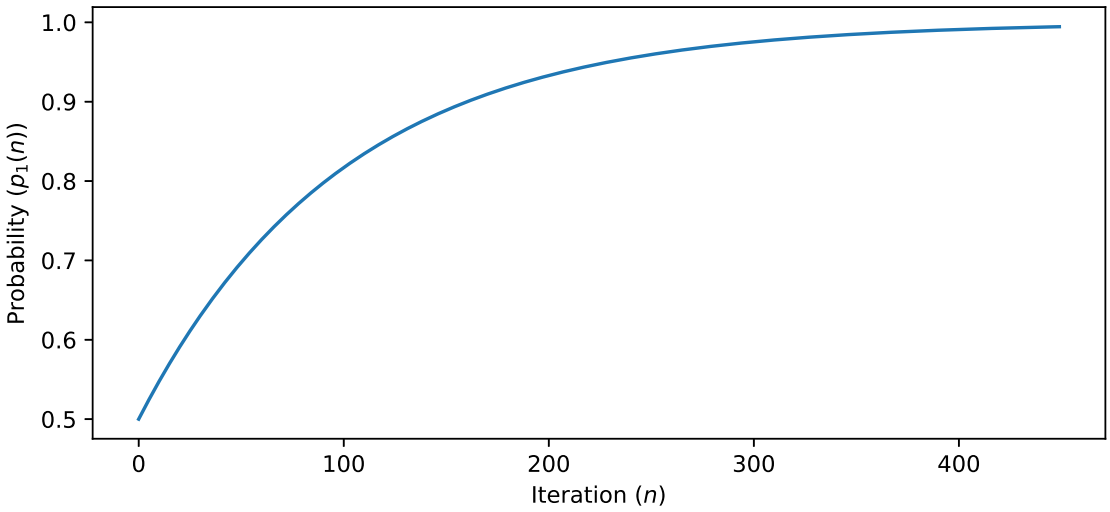


Figure 3.1: An example of the probability development for a continuous LA.

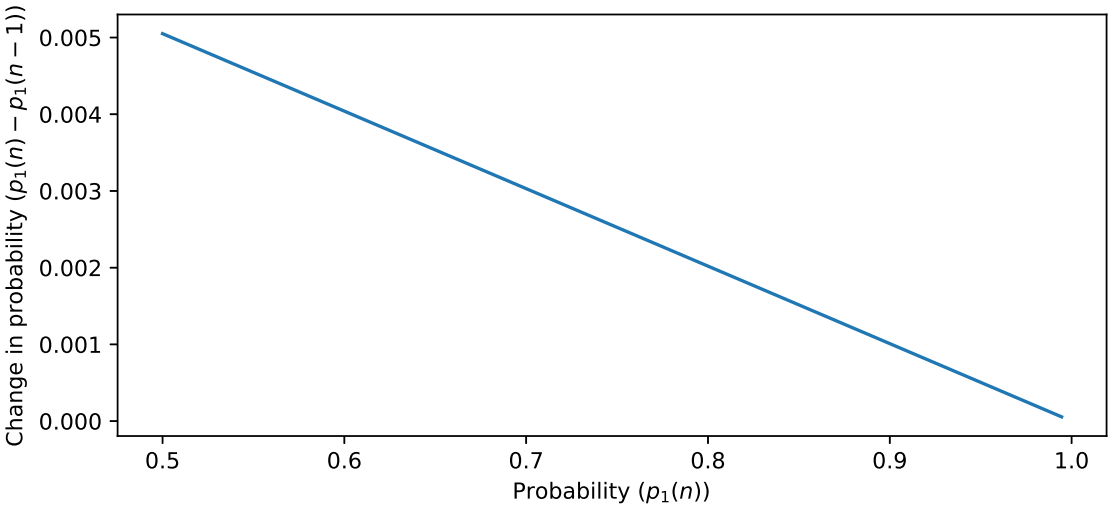


Figure 3.2: An example of the *change* in probability for a continuous LA.

type. The next quantum improvement to speed and efficiency was achieved by the invention of discretizing the action probability space. Unlike the continuous type of VSSA, where the probabilities can attain any value in the closed interval $[0, 1]$, the discretized type of VSSA can only attain a *fixed number* of values in the closed interval $[0, 1]$. More specifically, the probabilities of discretized VSSA for the two-action case can only attain the starting probability plus or minus an integer number times the learning parameter. For the multi-action VSSA, the learning parameter is handled differently, as discussed later, but the probabilities still operate on a discretized space. Thathachar *et al.* [76] were the ones to introduce the family of *discretized* algorithms.

Discretized algorithms are realized by redefining the likelihood of choosing either of the actions in the solution space to only a fixed number of values, as explained above. Consequently, the updating of the probabilities themselves is done in steps, as opposed to being done in a continuous manner. In this way, the discretized algorithms can be viewed as a hybrid combination of both FSSA and VSSA. Thus, the discretized algorithms operate within a finite set, similar to the case of FSSA. However, their behavior is completely determined by the probability vector that is a characteristic of the VSSA. The benefit of discretizing the probability space is that the *learning* is achieved in a step-wise manner, mitigating the problem of diminishing probability changes as an action probability approaches unity. Thus, if the updating of probabilities is discretized, the change in probability upon a Reward/Penalty can remain constant throughout the learning process. The reader should note that the learning parameter can also be adjusted over time, leading to a non-linear discretized VSSA, where the probability values are unevenly spaced in the closed interval $[0, 1]$ [76, 77].

Let us consider the example of a simple discretized DL_{R-I} scheme. The operation of the DL_{R-I} resembles the rules for the L_{R-P} machine as described above for the continuous world. However, in the case of a Reward-Inaction scheme, only the rewards are processed, while the penalties are ignored. Let us denote Δ as the learning parameter, which is a common notation for the learning parameter in discretized VSSA. This discretized learning parameter is used in the updating function for adjusting the action probabilities based on the feedback from the Environment. In the case when we consider the two-actions case, the updating function of the automaton can be specified as follows [49]:

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

where the min and max operators are invoked to ensure that the probabilities remain within the $[0, 1]$ interval. Consequently, upon α_1 being chosen by the automaton and receiving a Reward from the Environment, its likelihood of being selected is increased with Δ. The likelihood of the other action is similarly decreased by the same quantity. The same principle applies to α_2 when it is chosen and rewarded. When the discretized VSSA scheme has more than two actions, the probability of

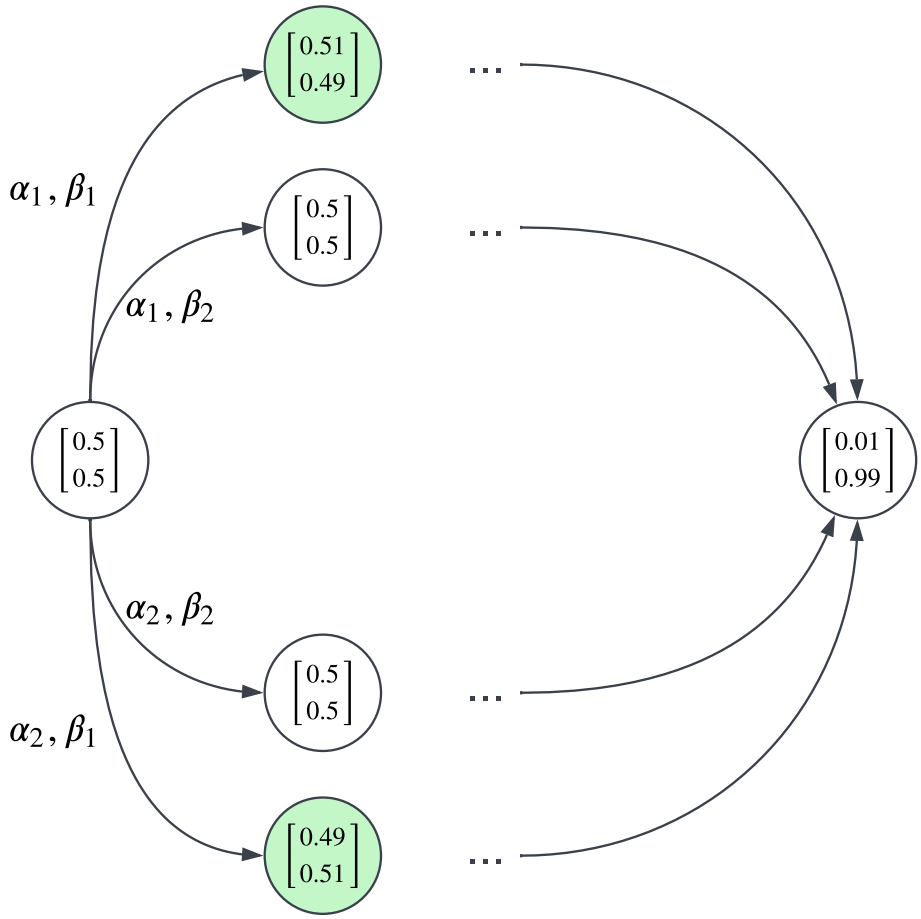


Figure 3.3: A visualization of a discretized DL_{R-I} concept.

each of the non-rewarded actions are decreased by $\Delta = \frac{1}{RH}$ (or made to zero), where H is a positive integer, and the rewarded action is adjusted such that the sum of the action probabilities remains unity.

As opposed to the case of a Reward, when an action is chosen and receives a Penalty from the Environment, no updating occurs, defined as follows:

$$\begin{aligned} &p_1(n+1) = p_1(n), \\ &p_2(n+1) = p_2(n), \\ &\text{for } \mathcal{A}(n) = \alpha_1 \text{ or } \alpha_2 \text{ and } \mathcal{B}(n) = \beta_2 = 1, \end{aligned}$$

which means that the penalties are "ignored". Understandably, the concept of the DL_{R-I} scheme is quite different from the L_{R-P} scheme as described earlier. We visualize the concept of the DL_{R-I} scheme in Figure 3.3. In the figure, we have an initial action probability, and the outcomes of different events can be interpreted from the arrows. Thus, when α_2 is chosen, and it receives a Reward (β_1), the probability vector becomes $P(n) = [0.49, 0.51]$, as the probability of α_2 is increased by $\Delta = 0.01$, where the learning parameter is specified for the particular problem. With a large Δ, the automaton will learn fast, but less accurately. However, with a smaller Δ, the automaton will be more likely to attain to an optimal solution. Consequently, the tuning of Δ is a trade-off and leads to a balance between the accuracy and the efficiency. On the right side of the figure, we have a circle that represents the action probability vector as the VSSA has *converged*. The concept of convergence in VSSA will be addressed below.

3.1.3 Estimator/Pursuit Algorithms

The invention of the discretization of the probability space made the VSSA algorithms more effective in terms of the number of iterations required before *convergence*, especially for experiments requiring a high accuracy. However, what we refer to as the next major discovery in the paradigm of VSSA, i.e., the EAs, represented possibilities for an even better speed and accuracy. Thathachar and Sastry presented EAs as a novel category of algorithms in [9]. As the name suggests, these algorithms are based on the concept of *estimation*. More specifically, in addition to the action probability vector, these algorithms maintain a *reward-estimate vector*. The reward-estimate vector keeps track of the different actions' probabilities of receiving a Reward. These estimates are further utilized in the updating function of the automaton.

By providing the machine with a new level and type of information, it can make even more informed decisions in its operation, and attain convergence faster when compared with non-estimation VSSA schemes. More specifically, the EAs have a property that makes them quantum steps faster because of the intertwined combination of the convergence of the reward estimates, $\hat{D}(t)$ (the notation is explained in more detail below), and the action probabilities themselves. If the actions are sampled *enough number of times*, their estimates will converge to their true underlying values (and the same principle applies to the actions themselves in terms of

the action probabilities). When the automaton selects actions based on the action probability vector, the estimates are both directly and indirectly influenced. At the same time, the action probability vector and reward-estimates vector are separate entities, meaning that the estimates can indicate that another action, other than the one that has the highest action probability, can possess the highest reward estimate. In this way, the information about the estimates can intelligently inform the automaton about the current *preferable* actions.

The properties referred to above imply that, asymptotically, the ranking of the estimates will be aligned to the ranking of the true reward probabilities with a high probability. Based on this property, we can reward the action (or actions) that has (have) the *best reward-estimates*, instead of the action that was chosen at any given time instant. In this way, the action probability of the action/actions with the best estimate is increased upon a Reward, regardless of which action *actually* received the Reward from the Environment. The estimates will be updated according to the true interactions with the Environment, i.e., the reward-estimate for the action that *actually* was chosen and received the Reward is increased in the reward-estimate vector. However, the likelihood of the action/actions with the highest reward estimates are increased in the action probability vector. The updating functionality of these algorithms is different from the previously-described ones, and this significantly reduces the number of iterations required before convergence. The reason for the EAs' superiority is that the probability of the inferior actions converging to zero is rendered faster, and consequently, they are also not sampled so often.

In the paradigm of EAs, we refer to a sub-type of them as *Pursuit* algorithms. These algorithms distinctly utilize the reward estimates. Thus, we say that they *pursue* the action that the automaton currently perceives to be the optimal one based on the reward-estimate vector. Thathachar and Sastry proposed the first pursuit algorithm, namely, the Continuous Pursuit Reward-Penalty (CP_{RP}) algorithm (discussed in [9], and referred to as the "Thathachar and Sastry's Estimator" algorithm). In this version, the currently perceived "best action" is always rewarded, and the "less optimal actions" are penalized. We also have a similar Reward-Inaction scheme, the CP_{RI}, as described in [12].

As a natural next step in the development of VSSA, researchers developed the combination of EAs incorporated with the discretization of the probability space. More specifically, researchers discovered the benefit of combining the discretized versions of VSSA and the Pursuit phenomenon. By the works of Lanctôt and Oommen, the "Thathachar and Sastry's Estimator" algorithm was enhanced in [9]. Thus, the authors proposed different schemes where the updating in probabilities was determined based on whether the reward estimate of the chosen action was smaller or larger than the other reward estimates. The authors also addressed and significantly improved the Pursuit algorithm with the discoveries presented in [78], referred to as the Generalized Pursuit Algorithm (GPA). Later, in [12], Agache and Oommen presented their Discretized Generalized Pursuit Algorithm (DGPA), discretizing their previously-presented GPA and essentially also generalizing the algorithms presented in [9]. In general, these algorithms have been referred to as Discrete Estimator

Algorithms (DEAs) in the Literature [78]. To differentiate between the estimator-based discretized algorithms and the Pursuit-based discretized algorithms, the term, DPA [2], has also later been used.

The reward-estimate vector is often denoted as $\hat{D}(t) = [\hat{d}_1(t), \ldots, \hat{d}_R(t)]^T$ in the Literature, and the corresponding state vector as $Q(t) = \langle P(t), \hat{D}(t) \rangle$[3]. The notation refers to the reward estimates of the actions in a general manner, where there are R actions in total. As mentioned above, the utilization of this vector in the automaton's operation can be achieved in different ways. In addition, calculating the estimates can also be done in different ways. More specifically, the estimates can be calculated using a Maximum likelihood scheme or a Bayesian approach [79].

The proofs related to the convergence of EAs are more complicated than those for traditional LA because of the complicating factor of ensuring that all actions, including the sub-optimal ones, are sampled enough number of times. The authors in [78] showed that the DEAs possess the so-called *Moderation* and *Monotone* Properties. Indeed, some earlier proofs for the related DPA algorithms were flawed and rectified in [11] and [14], respectively.

3.1.4 Hierarchical Algorithms

The action probability vector, $P(n)$, is normally initialized with each of the action probabilities being equal to $p_i(0) = \frac{1}{R}$, where $i \in \{1, 2, 3..., R\}$. As R increases, the likelihood of the automaton choosing one of the actions is correspondingly decreased. If we compare the initial probabilities when $R = 10$ and $R = 1,000$, the probability that the LA selects α_1 in the first alternative is 100 times more likely when compared with the second alternative. Consequently, the more actions we have in a problem, i.e., the larger the solution space we have, the less significant a single action becomes.

The most important aspect of VSSA concerning the convergence to the optimal action is that all actions need to be explored sufficiently. Let us consider the case when we have a VSSA scheme with 100 actions. Unless the learning parameter is sufficiently small, we might explore only half of the actions, e.g., the latter 50 actions. The explored actions might have been rewarded, making the probability of selecting the remaining 50 first actions correspondingly smaller, and the optimal action can be one of the unexplored actions. Although a small enough learning parameter can ensure that we explore all actions sufficiently, as time goes to infinity, it might result in an impractical time frame before convergence is achieved in practical scenarios.

Due to the aforementioned "problem of a large solution space," it was imperative that VSSA should be capable of supporting larger solution spaces. Incorporating "structure" into the ordering of the actions represents a viable solution to problems with a *large* number of actions, when this set is relatively high. The invention of incorporating structure into VSSA is the most recent advancement in the field. The

[3]The $Q(t)$ referred to here should not be confused with the query at time instant n ($Q(n)$) in the OMA related contexts in this dissertation.

authors of [2][4], were motivated to devise a scheme by which small subsets of actions (for example, of cardinality two) were compared, and the result of their comparison was trickled up a structure to avoid dealing with an R-action LA directly. They referred to their proposed solution as the HCPA. More details about this concept are presented in detail below[5].

The HCPA is based on the concept of a hierarchy of LAs, where the actions that interact directly with the Environment are located at the leaves of a binary tree structure [2]. The HCPA consists of CPAs, ordered in levels, where the LAs are connected in a parent-child organization. More specifically, we have a root node at the top of the tree structure. All nodes in the tree have a set of possible actions, and these actions represent the possible nodes at the next level. In the HCPA [2], all the nodes have two possible actions[6]. Thus, they can activate the child on its left or right branch. The nodes at the second lowest level, i.e., the level above the *leaves* of the tree, are the automata that ultimately choose the action to interact with the Environment. Thus, the action chosen as output is selected through the path from the root node to the arbitrary *leaf action*.

Because the individual CPA in the HCPA tree structure has to only handle two actions, the authors avoided the problem of dealing with insignificant probabilities in the action probability vector. The HCPA was many orders of magnitude faster when compared with other legacy types of LA [2]. The significant improvement in speed and accuracy was achieved because the HCPA combines the concept of VSSA, EAs, and the novel structure in the LAs operation. In order for the HCPA to *learn* the optimal action, it has to find the *optimal path* throughout the tree structure. The learning is achieved at every level, and at every node of the tree, by rewarding the actions on the reverse path of the rewarded action. Because the HCPA utilizes the Pursuit phenomenon, the currently-estimated "best" action is rewarded upon a Reward received from the Environment, even if another action triggered the Reward. In order to accomplish global learning, each individual LA is responsible for doing the learning locally, and the results of this learning are, thus, trickled up in a recursive way by examining just a pair of LA at a time.

To better understand the concept of the HCPA we present in Figure 3.4 a step-wise visualization of its concept for choosing the action to interact with the Environment. As explained above, the HCPA is based on the realization of a hierarchical tree structure with a root node. Step 1 in the action selection process involves uniform random sampling as per the root node's action probability vector. In the

[4]In principle, the idea of incorporating structure into LA is not something completely new. Already in 1994, with the works of Papadimitriou [19], there was an attempt to hierarchically structure the automatons in a tree-like manner. The difference between the works of Papadimitriou and the HCPA is described in detail in [2].

[5]The reader should note that only a brief explanation is presented here in order to avoid repetition because the finer details about the HCPA are included in the research papers within this dissertation.

[6]Each of the CPA nodes can have more than two actions. However, because we are trying to mitigate the diminishing action probability as R becomes large, we should keep the number of actions at each node to a minimum.

example, the left branch is selected. At the next level, the first LA is activated, and by a new uniform random sampling as per its action probability vector, its right branch is selected. As we can see from the figure, in Step 4, the leaf action, number four, is activated and chosen to interact with the Environment.

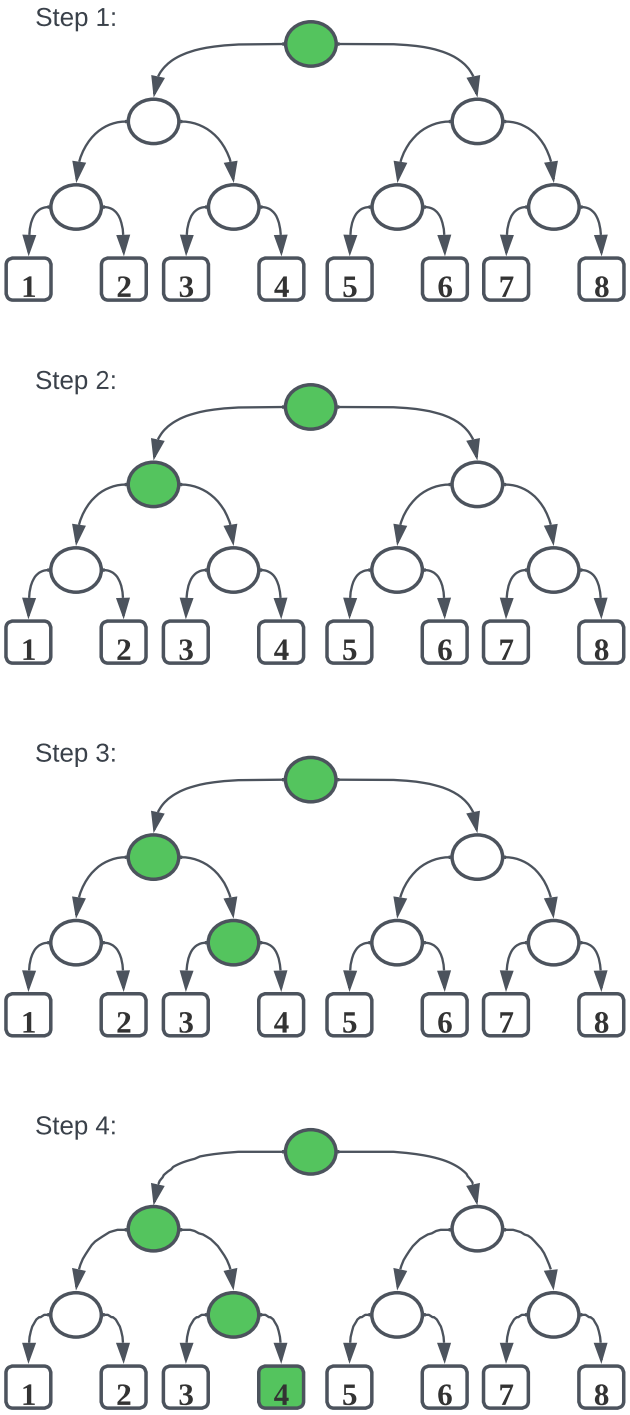


Figure 3.4: Step-vise visualization of the path throughout a hierarchical structure in the HCPA.

3.2 VSSA and Convergence

The VSSA schemes have a different convergence criterion when compared to the FSSA schemes. For FSSA schemes, we have states, and when the LA is in a certain state, it will select the corresponding action as the output linked to that state. Consequently, the convergence is related to the certainty of the automaton, and the state depth can be adjusted to accommodate the accuracy requirement for the specific problem. In the case of VSSA, the machine's learning and the *memory* are maintained in its action probability vector. Consequently, for the VSSA schemes, convergence is defined in terms of the components of the probability vector. Specifically, for VSSA, we say that the automaton *has converged* once any component of the action probability vector is equal to, or above, a certain threshold. This threshold is referred to as the "convergence criterion".

The convergence criterion for the VSSA scheme is normally set to be an arbitrary value close to unity [50]. As the action probability of an action increases, so does the likelihood of that action being chosen by the automaton. When the probability of any of the actions in the solution space of the machine is close to unity, the automaton will, traditionally, tend to *only* choose that action. In this case, we say that the automaton has *converged* to an action. The convergence criterion is connected to the efficiency of the algorithm. Thus, in many cases, a convergence criterion of, e.g., 0.95 or 0.99 is sufficient for assessing the machine's performance, considering that the automaton is quite certain about its favored action. The convergence criterion is a method of specifying how certain we want the automaton's solution to be.

Another question to be considered is whether the automaton has converged to the optimal action. In practical cases, we need to learn the optimal action. However, when we know which action is truly the optimal one, as we do in a simulation environment, we can assess the LA's *accuracy*. Because the VSSA algorithms operate in stochastic environments, their performance is typically evaluated based on the average performance over several experiments, typically referred to as the "ensemble of experiments". The convergence criterion is our method of assessing how certain the decision of the automaton is. However, even if we have a high convergence criterion, often referred to as a high accuracy requirement, it might not necessarily mean that the algorithm is, in fact, accurate. If the learning parameter is not small enough, the automaton might be unable to find the optimal action.

We emphasize the importance of the reader's understanding of the relationship between the various parameters encountered within the LA paradigm. By doing this, one can appreciate the mechanisms invoked to improve these algorithms. For example, continuous VSSA have slow convergence because the changes in probability become less as any probability approaches unity. The discretized VSSA do not suffer from the same impediment. However, if we compare them using a lower convergence criterion, e.g., 0.85, we see that the difference between their efficiency might be obscure as if we compared them using a convergence criterion of 0.99. Another example involves choosing a large learning parameter. A large learning parameter will ensure faster convergence compared with a smaller learning parameter, but the

solution of the automaton might not be correct. Thus, the learning parameter, the convergence criterion, and the accuracy are all related, and influence one another.

3.3 Chapter Summary

In this chapter, we have informally explained the concept of VSSA and highlighted why these algorithms are distinct from the FSSA algorithms described in the preceding chapter. We have briefly explained all the discoveries and improvements proposed within this domain throughout the years. The discovery of VSSA increased the efficiency of LA compared to its FSSA counterpart. The initial VSSA varieties were continuous. However, their convergence was slow when the accuracy requirements were high. A subsequent discovery, namely that of discretizing the probability space, was not hindered by the same obstacle. The discovery of Estimator/Pursuit algorithms was another significant development in the field of LA. The algorithms can outperform other LA algorithms by estimating the reward probabilities and including this knowledge in the automaton's updating function. The most recent innovation is the implementation of *structure* into LA, which enables the exploration of much bigger solution spaces. These discoveries serve as the foundation for the subsequent enhancements proposed in this dissertation.

Chapter 4

Novel FSSA Solutions

As explained earlier, the FSSA type of LA can be used to solve partitioning problems. Within FSSA, we have the family of OMA algorithms that are tailored to solve such problems. By representing the groups as the *actions* in the LA and distributing the entities to be grouped as *abstract objects* that traverse the states of the automaton, the OMA can be used to obtain a grouping of entities and also monitor them over time. One of the key strengths of the OMA is its ability to handle highly stochastic environments [38, 75]. Additionally, the OMA algorithms are based on *queries* presented as inputs to the automaton, where a query consists of abstract objects presented together under the assumption that their joint accessibility ultimately implies that they should be grouped. These queries can be presented over time, and the OMA can handle problems without initially having any understanding of the intricacies of the problem. Consequently, the OMA requires no information up-front[1], unlike other traditional clustering methods like, e.g., K-Means.

The reader may question why VSSA are not suitable for partitioning problems when VSSA, in general, are more effective than the FSSA type of LA. To respond to this question, we mention that two types of solutions are possible with VSSA. The first approach could be by representing each abstract object by means of a separate LA, where the LA is rewarded when the queried objects' LAs output the same action. Likewise, they are penalized when the queried objects do not output the same action, and thus, we would only trigger outputs from the queried objects. Still, as we know from VSSA, their output is based on a probability vector, making them likely to try other actions than the currently "best action" according to the sampling of the action probability for selecting the output action. This approach is cumbersome because each entity to be grouped requires an action probability vector (and a reward estimate vector in the case of an EA being utilized), and it would require each LA to converge. This, in turn would require many more iterations than having the objects traverse the states of an FSSA with, e.g., six states per action. More groups and items would also increase the complexity of the underlying LA. On the other hand, we believe that this approach could be a research direction worth exploring, although this avenue has not been a part of our current study.

[1] In general, the only information that the OMA requires upfront is the number of groups, their sizes, and a list of the abstract objects to be grouped. More details about the characteristics of the OMA will be provided later, as we present our innovations to the field.

The second approach for using VSSA to solve partitioning problems could be by representing all possible grouping configurations as different actions in the automaton. Thus, we represent the different solutions as actions. However, this would be a non-scalable solution. Even with as few as four objects to be grouped into two groups of size two, we would have six components in the action probability vector. Additionally, an extremely difficult question to resolve in this case would be that of determining how to solve the problem around rewards and penalties when you have query pairs, and a single pair would be in several of the configurations, while at the same time, there would be objects in the configuration that should not be rewarded. Therefore, as the reader can understand, VSSA are not suitable for solving partitioning problems.

Establishing groups based on some underlying and, common, completely-unknown criterion, is a complex and highly challenging task, and the problems are NP-hard [38]. The reason for the complexity is that the number of potential solutions and permutations of objects increases significantly with the number of objects in the partitioning problem[2]. As presented earlier, the OMA algorithms can solve partitioning problems. However, they have, until the works presented in this dissertation, substantial drawbacks that limited their applicability to real-life problems. One of these prominent limitations was that the algorithms could only handle problems of equally sized groups. In other words, the OMA algorithms could previously only handle EPPs. In this dissertation, we have made substantial research efforts in the field of enhancing the power of OMA. Indeed, we have made novel contributions to using FSSA-based algorithms to solve more complex grouping problems. More specifically, we have proposed substantially better solutions to the algorithms in the OMA family. In this chapter, we will address these novel contributions.

4.1 Proposed OMA Algorithms

As alluded to earlier, the OMA algorithms are powerful LA-based schemes for solving partitioning problems. Four variants exist in the family of OMA algorithms, namely the Vanilla OMA, the EOMA, the PEOMA, and the TPEOMA. The Vanilla OMA suffered from the Deadlock Situation, and the EOMA presented a solution that mitigated this issue. With the PEOMA variant, the pursuit concept was incorporated through pursuing the *better* queries by estimating the joint accessibility of the abstract objects and filtering out queries that are less likely to be accessed together. The PEOMA is especially beneficial when we have highly noisy systems [17, 18]. The TPEOMA variant implements the transitivity concept. More specifically, we can utilize the estimations established in the PEOMA structure to infer other (possibly non-accessed) likely query pairs when presented with a query from the system. In this way, the TPEOMA represents an especially beneficial approach when we have sparse information from the system [74]. The TPEOMA is the most advanced ver-

[2]To avoid repetition, we do not detail the complexity of these problems here. The corresponding complexity analysis is addressed in the articles included in this dissertation.

sion of the OMA algorithm currently in use. The EOMA, on the other hand, serves as the foundation for both the PEOMA and the TPEOMA. As a result, the EOMA variant has been the primary center of our attention throughout this dissertation. Due to the fact that the proposed modifications are suggested to the fundamental underlying OMA structure, the enhancements that have been subsequently proposed can be implemented with any of the existing OMA variations.

Within the OMA paradigm, the objects that are *accessed together* are presented to the automaton as a *query*. The reader might ask what triggers the *"accessed together"* property. Only our creativity limits the type of grouping problems that we can model and monitor, using OMA algorithms. As an example, we can trigger a query of abstract objects representing files that are accessed together on a server. Or a query can be generated once two supermarket items are put into the same shopping cart. A query can also be triggered after checking whether a criterion of similarity is satisfied, i.e., if the signal properties of two radio towers measured by a correlation function, is within a certain percentage of the difference between the least and most similar signal patterns. As we understand, the partitioning problems that the OMA can solve are versatile. However, the existing OMA variants have limitations to the problems that they can solve.

The ability to only be able to handle equally-sized groups, i.e., EPPs, is a limitation that is shared by all of the existing OMA algorithms. When faced with real-life challenges, we may encounter issues that involve groups of unequal sizes. Consequently, to enhance the field, we need to discover (or rather "invent") new ways to instruct the OMA algorithms so that they can also solve NEPPs. If we consider a grouping example where we want to distribute ten files across two databases, and one of the databases has room for three files, and the other has room for seven files, the existing OMA algorithms will be incapable of solving the problem. Only if the databases were equally-sized, i.e., the databases had room for exactly five files each, could the existing OMA versions solve it. This example illustrates that real-world applications often involve varying partition sizes and capacities. Traditional OMA algorithms, which rely on equally-sized groups in partitioning scenarios, lack the flexibility to effectively accommodate real-life scenarios. The inability to handle unequal partition sizes limits the applicability of existing OMA algorithms in diverse domains, including resource allocation, logistics, and those in operations research.

Our first proposed OMA scheme is the Greatest Common Divisor (GCD) OMA variant in Section 4.1.1. This variant still imposes a constraint to the partition sizes that it can handle. More specifically, it can only tackle problems where we have a non-unity GCD between the groups in terms of their sizes. The second proposed OMA scheme is the Partition Size Required (PSR) variant. Thus, the GCD OMA can handle NEPPs, but a size requirement to the partitions remains. The PSR OMA, discussed in Section 4.1.2, can handle partitions of arbitrary sizes. Indeed, the PSR OMA is able to handle more general OPPs. More specifically, the PSR OMA can handle both EPPs and NEPPs. Both these algorithms significantly advance the capabilities of OMA, and open avenues for new opportunities for solving complex real-world problems. The proposed GCD and PSR variants mark a signif-

icant breakthrough in the field of OMA. By addressing the challenges of unequal group sizes, and the absence of size requirements, these algorithms offer unprecedented flexibility and efficiency in solving complex grouping problems. As a result, they pave the way for innovative applications and research opportunities in this domain, and they can also inspire innovations in other LA-based domains.

The GCD and the PSR variants are compatible with all the earlier existing OMA algorithms. Because of this, the name of the variation of the newly-suggested techniques, will be associated with the fundamental OMA type, so as to make the documentation easier to understand. Consequently, the nomenclature for the GCD variations generates the terms GCD-OMA (Vanilla OMA), GCD-EOMA, GCD-PEOMA, and GCD-TPEOMA when combined with any member of the OMA family. In a similar fashion, when one invokes the PSR, one would get the PSR-OMA (Vanilla OMA), the PSR-EOMA, the PSR-PEOMA, and the PSR-TPEOMA. When we talk about algorithms in a more general sense, we will refer to them using a generic terminology such as GCD OMA and PSR OMA, respectively.

4.1.1 The Greatest Common Divisor (GCD) OMA

The OMA is built on the principle of gathering the objects that are frequently accessed together into groups, with the fundamental assumption that objects that are accessed together, typically, pertain to the same underlying partition. This serves as the scheme's conceptual basis, and it remains as the most important principle in the GCD OMA as well.

The existing OMA algorithms intelligently replace an object that transitions between actions (partitions), ensuring that all groups maintain an equal number of objects at any given time instant. This functionality prevents all objects from converging to the same action. As an example, if two complete groups, that, in reality, should be in different groups, reside within the same partition, both of these groups will be consistently rewarded and the need for them to be disintegrated disappears. Thus, the queries with object pairs from either of these groups, will be rewarded, because they are in the same action. However, they should optimally be two separate entities. The *switch-action-and-replace* behavior of the OMA prevents this undesirable behavior from happening. However, with the GCD OMA variant, we resort to a solution where we extend the states onto a larger state space, while maintaining the switch-action-and-replace phenomenon that is a fundamental attribute of the already existing OMA algorithms.

In the GCD OMA, we broaden the states of actions into a larger state space by creating additional sub-partitions for each action (state expansion), and we virtually connect these sub-partitions. In this way, the automaton sees one or more sub-partitions as an overall partition, even if these sub-partitions are, in fact, different actions within the automaton. This proposed variant, allows the OMA to handle NEPPs. However, we still have a size requirement to the partition sizes. More specifically, the GCD OMA operation necessitates a shared non-unity GCD among the partition sizes. This means that the GCD OMA can handle NEPPs with a

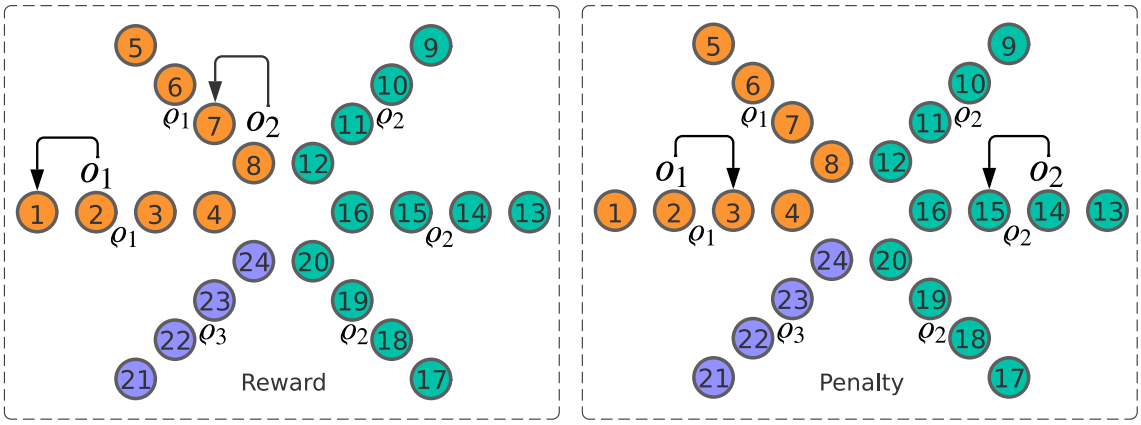


Figure 4.1: Example of the extended state-space of the GCD OMA upon a Reward and a Penalty.

non-unity GCD, and that the partition sizes have a relation between themselves.

The GCD OMA has one or more sub-partitions that constitute an overall partition. This behavior is achieved by the state expansion phenomenon explained above, and by linking sub-partitions together. More specifically, in the GCD OMA variant, we utilize the GCD as the number of objects that can reside within a sub-partition. Consequently, the GCD variant can handle all group size combinations composed of a multiple of the GCD. Thus, if we have three groups, where the size of the first group is three, of the second is nine, and of the third group has room for twelve objects; our GCD is three. We can then activate a single sub-partition for the first group as an overall partition. In the second group, we must invoke three sub-partitions as an overall partition. Lastly, we need four sub-partitions for the third group's overall partition. The rewards, penalties, and other GCD OMA functionalities need to be carefully adjusted according to the described concept, as described in more detail presently.

To construct the baseline for the GCD OMA, we need to determine the GCD between the group sizes. We denote the GCD as $\Lambda > 1$, and this parameter needs to be determined for the particular grouping problem we are dealing with. Based on Λ, we can establish the links needed to fulfill the overall size requirements for the groups. We consider the sub-partitions that are linked as a single entity within the automaton. In this way, if a group size is greater than Λ, we need to consider more than one sub-group in the overall partition. All of the actions in a GCD OMA need to have Λ objects, and the number of sub-partitions that needs to be linked into an overall partition ϱ_k, where $k \in \{1, 2, ..., K\}$, is given by $x_k = \frac{\rho_k}{\Lambda}$, and ρ_k, $k \in \{1, ..., K\}$, is the number of objects in each partition. When we have a partition ϱ_k which has a size equal to Λ, it follows that $x_k = 1$, which means that only a single sub-partition is adequate to accommodate its group size. Thus, when $\rho_k = \Lambda$, no links are needed for the group ϱ_k.

In order for the automaton to handle the partitions that now are extended onto a larger state space, the automaton needs to consider the corresponding *state ranges*. In every sub-partition, we have room for Λ objects. Each of the sub-partitions

(actions) have S states, and x_k indicates the number of sub-partitions that have to be linked together to fulfill the size of ϱ_k. Furthermore, the state range of an overall partition, ϱ_k is given by:

$$\iota_k = \{\max(\iota_{k-1}) + 1, ..., \max(\iota_{k-1}) + x_k S\}, \ \forall k, \tag{4.1}$$

where ι_k represents the state range for partition ϱ_k. We emphasize that partition $k = 1$ has no previous partition, meaning that the first partition considered in the GCD OMA is $\iota_1 = \{1, 2, ..., x_k S\}$. When constructing the state range for the subsequent partitions, the max function in the preceding equation indicates that the highest value from the state range of the previous partition should be used as the starting point for the next state range.

To understand the concept of the state ranges, we consider the following example. If we have $S = 5$, and $x_k = 1$ for the first partition, the first overall partition has the state range $\iota_1 = \{1, 2, ..., 5\}$. Consequently, the maximum of the first state range is 5, and it follows that the state range of the second overall partition is $\iota_2 = \{6, 7, ..., 20\}$ when the partition has $x_k = 3$. Furthermore, it is important to notice that the number of sub-partitions (actions) in the automaton, R, is given by $R = \sum_{k=1}^{K} x_k$. In order to correctly comprehend the algorithms, we repeat here that the current state of an object o_i is denoted by ϕ_i, where $i \in \{1, 2, ..., O\}$.

To implement the GCD OMA variant when it concerns the other algorithms within the OMA paradigm, we only need to propose modifications to the baseline algorithms. More specifically, the innovations can be implemented in the OMA or the EOMA variant, which provides the fundament for the PEOMA, and the TPEOMA, respectively. In this dissertation, we refer to the GCD OMA variants as the GCD-OMA (Vanilla OMA), the GCD-EOMA, the GCD-PEOMA, and the GCD-TPEOMA. Further, since the EOMA is preferred over the OMA variant, we only describe here the GCD-EOMA implementation. However, the GCD concept can easily be extended to the Vanilla OMA variant.

The algorithmic description of the GCD-EOMA is presented in Algorithm 4 [75]. As we can perceive from the description, the Reward functionality utilized is described in Algorithm 6, which implies that the EOMA Reward is not modified from its EOMA version. We describe the Penalty operation of the GCD-EOMA in Algorithm 5.

As in Algorithm 4, we initialize the objects into the automaton such that we distribute the objects uniformly random across the R boundary states available. Unlike the traditional EOMA, where $\frac{O}{K}$ objects are distributed into each of the actions, we now place Λ objects in each of the sub-partitions (actions). As the queries are presented to the automaton, the rewards and penalties are processed according to the policies described in Algorithms 6 and 5, respectively [75]. In the GCD-EOMA, a Reward is presented as the output from the Environment if the objects in the query are currently within the same *state-range*, and a Penalty is presented if they are not currently in the same *state-range*. Hence, the overall partition is considered, instead of merely considering whether the objects are in the same action or not.

Algorithm 4 GCD-EOMA

Input:

- The objects $\mathcal{O} = \{o_1, ..., o_O\}$, and S states per sub-partition.
- A sequence of query pairs, denoted as Υ, where each entry $Q = \{o_i, o_j\}$.
- Initialized ϕ_i for all objects. Initially all ϕ_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 any of the two most internal states, and the converged partitioning is then 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.
- ϕ_i is the state of o_i and is an integer in the range $\{1, 2, ..., RS\}$.
- If $\phi_i \in \iota_k$, 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 $k \in \{1, 2, ..., K\}$.

```
1: while not converged or |Υ| queries not read do
2:     Read query Q = {o_i, o_j} from Υ
3:     if φ_i and φ_j ∈ ι_k then          ▷ If the objects are in the same state range
4:         EOMA Process Reward (Algorithm 6)
5:     else                               ▷ If the objects are in different state ranges
6:         GCD-EOMA Process Penalty (Algorithm 5)
7:     end if
8: end while
9: Output the final partitioning based on φ_i, ∀ i.   ▷ According to the state ranges
```

In order for the reader to better understand the GCD-EOMA concept, let us consider the example visualized in Figure 4.1. In this example we have $S = 4$, and $\rho_1 = 12$, $\rho_2 = 18$ and $\rho_3 = 6$. In this scenario, we need to consider links in the GCD-EOMA operation, because the group sizes exceed $\Lambda = 6$, which is the case for this problem. From the figure, we can observe that the first state range is defined as $\iota_1 = \{1, 2, ..., 8\}$, and it represents the overall partition of ϱ_1. Further, $\iota_2 = \{9, 10, ..., 20\}$ is the overall partition of ϱ_2, and $\iota_3 = \{21, 22, ..., 24\}$ is the overall partition of ϱ_3. Additionally, we visualize the operation upon a Reward and a Penalty for a query, $Q = \{o_1, o_2\}$. In the example on the left side, the queried objects are inside the same state range, and they are consequently rewarded. In the example on the right side, the queried objects are in different state ranges, and they are, consequently, penalized.

Algorithm 5 GCD-EOMA Process Penalty

Input:

- The query $Q = \{o_i, o_j\}$, and the states of the objects in Q ($\{\phi_i, \phi_j\}$).

Output:

- The next states of o_i and o_j.

if $\phi_i \bmod S \neq 0$ and $\phi_j \bmod S \neq 0$ **then** ▷ Neither are in boundary
    $\phi_i = \phi_i + 1$
    $\phi_j = \phi_j + 1$
else if $\phi_i \bmod S \neq 0$ and $\phi_j \bmod S = 0$ **then** ▷ o_j is in boundary
    $\phi_i = \phi_i + 1$
    $temp = \phi_j$ ▷ Store the state of o_j
    o_l = *unaccessed* object in group of *staying* object (o_i) closest to boundary
    $\phi_j = \phi_i$
    $\phi_l = temp$
else if $\phi_i \bmod S = 0$ and $\phi_j \bmod S \neq 0$ **then** ▷ o_i is in boundary
    $\phi_j = \phi_j + 1$
    $temp = \phi_i$ ▷ Store the state of o_i
    o_l = *unaccessed* object in group of *staying* object (o_j) closest to boundary
    $\phi_i = \phi_j$
    $\phi_l = temp$
else ▷ Both are in boundary states
    $temp = \phi_i$ or ϕ_j ▷ Store the state of *moving* object, o_i or o_j
    $\phi_i = \phi_j$ or $\phi_j = \phi_i$ ▷ Put *moving* object and *staying* object together
    o_l = *unaccessed* object in group of *staying* object closest to boundary
    $\phi_l = temp$ ▷ Move o_l to the old state of *moving* object
end if

The OMA was developed to organize objects in such a way that those that are accessed often are grouped together under the presumption that these objects pertain to the same underlying partition. This notion is also essential to the GCD's overall concept. The current OMA algorithms make use of a behavior known as "switch-action-and-replace" in order to stop objects from inaccurately moving toward the same action, and to also keep the size of each group to be the same. This same phenomenon is built upon in the GCD OMA, which creates extra sub-partitions for each partition and then virtually connects those sub-partitions to one another in order to manage NEPPs. However, the GCD OMA requires for a common, non-unity, GCD among the various partition sizes. This, in turn, means that it can only deal with NEPPs that have a non-unity GCD and a consequent relationship between the various partition sizes.

Algorithm 6 EOMA Process Reward

Input:

- The query $Q = \{o_i, o_j\}$.
- The states of the objects in Q ($\{\phi_i, \phi_j\}$).

Output:

- The next states of o_i and o_j.

1: **if** $\phi_i \bmod S \neq 1$ **then**
2: $\quad \phi_i = \phi_i - 1$ ▷ Move o_i towards the innermost state
3: **end if**
4: **if** $\phi_j \bmod S \neq 1$ **then**
5: $\quad \phi_j = \phi_j - 1$ ▷ Move o_j towards the innermost state
6: **end if**

4.1.2 The Partition Size Required (PSR) OMA

The GCD OMA variant requires a size relation between the group sizes. Thus, the groups need to have a common non-unity GCD. Consequently, the GCD variant still has limitations to the types of NEPPs that it is able to solve. Our next enhancement is to develop an OMA scheme that is even more versatile to the problems that it can solve.

Our proposed PSR OMA variant can generally solve NEPPs without a common divisor between the group sizes, i.e., it can solve partitioning problems with arbitrary partition sizes. This means that the PSR OMA can solve OPPs, but we still utilize the notation of the EPP/NEPP to separate the innovations from the existing variants distinctly. The distinct behavior that differentiates the PSR OMA from the GCD OMA is that the PSR variant can adaptively swap the group sizes themselves within its operation. Thus, the PSR can change the number of objects in one action upon a certain object distribution, allowing it to adapt group sizes according to the incoming queries intelligently.

The PSR variant represents a major innovation to the OMA paradigm. However, it still has one requirement in relation to the partition sizes, which is that it requires the user to know the group sizes *a priori*. Thus, the partition sizes need to be pre-specified, i.e., the PSR OMA requires groups with pre-specified cardinalities. Initially, the PSR OMA distributes the abstract objects representing the items to be grouped into K actions. Thus, in the PSR OMA, we have the same number of actions as we have partitions (similar to all the existing OMA variants). The difference from the existing variants is that we can have a different number of objects inside each of the actions. In the PSR OMA, the actions in the OMA have the same number of objects as the group sizes in our partitioning problem itself. In this way, the PSR OMA is completely different from the GCD variant. While in the GCD OMA, we need to *link* sub-partitions to accommodate the group size requirement,

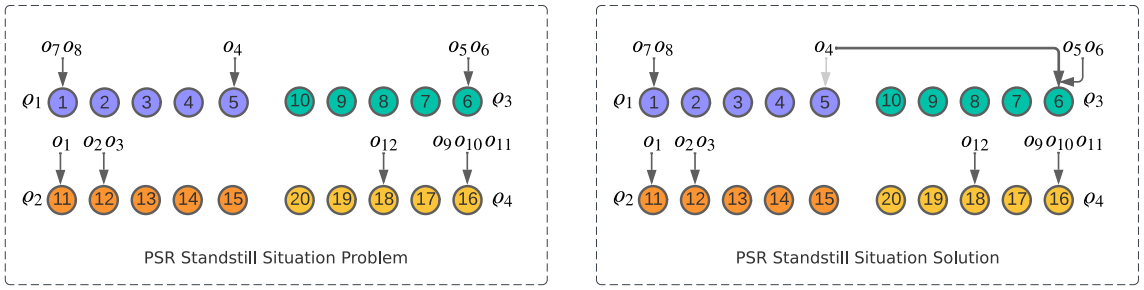


Figure 4.2: Example of a Standstill Situation and the PSR OMA's solution for it.

in the PSR OMA, the actions are designed to directly represent the group sizes.

The PSR OMA possesses problems that do not exist when we handle the EPP case that existing OMA algorithms can tackle. The same applies to the GCD OMA, because the GCD OMA continues to handle an equal number of objects in each action (partition). More specifically, we have the complicated issue that objects can *get stuck*, in what we refer to as a Standstill situation. The "Deadlock Situation" described in [73] must not be confused with the distinct "Standstill Situation". The occurrence of a "Standstill Situation" happens when a smaller partition is trapped inside of a larger partition. This can prevent the automaton from converging, and it can even result in an endless "infinite" loop of objects moving back and forth between actions and states. When a smaller partition is contained within a larger partition, objects are prevented from being grouped together, as they should be in Δ^*. Consequently, the problem of the Standstill Situation affects the convergence and performance of the machine. Thus, it can result in a slower convergence, or that the machine does not even converge within an infinite time frame, and even result in non-optimal partitioning solutions.

To understand the Standstill Situation in more detail, let us consider a simple example where we have four partitions. Two of the partitions have room for three objects, the next partition has room for two objects, and the last partition has room for four objects. Using the notations established earlier, we have $\rho_1 = 3, \rho_2 = 3, \rho_3 = 2, \rho_4 = 4$. We equip the automaton with five states ($S = 5$). In this problem, we have twelve abstract objects, and four partitions. After a series of queries, the automaton can then become stuck in a possible Standstill Situation, as visualized on the left side in Figure 4.2. Thus, the automaton is "stuck". Let us further assume that Δ^* has the following groups: $\{o_1, o_2, o_3\}$, $\{o_4, o_5, o_6\}$, $\{o_7, o_8\}$ and $\{o_9, o_{10}, o_{11}, o_{12}\}$.

Let us continue with the example of a Standstill Situation problem as visualized in Figure 4.2. In the case of a noise-free system, the object o_4 will be queried together with o_5 or o_6 as $Q = \{o_4, o_5\}$. If we only consider the existing OMA functionality, such a query, when presented to the automaton, will result in a Penalty. In a normal Penalty operation, o_4 is to be moved to another partition. If we consider that o_4 is moved to ϱ_3, o_5 will replace o_4 and is moved to state five. This operation does not bring o_4, o_5 and o_6 closer together. Actually, we are not any closer to the optimal solution than we were before we were given the query. The described process might continue until ϱ_1 is practically reset, meaning that all other objects are moved from

ϱ_1 to another group, so that there is enough room for o_4, o_5 and o_6 to be inside that particular partition. In the existing OMA, ϱ_3 was initialized with two objects, and it cannot change its partition size. Consequently, the Standstill Situation, without any modifications to the existing policies in the OMA paradigm, complicates the movement of the objects, and it can make convergence highly unlikely, i.e., within reasonable time frames. Therefore, in our PSR OMA variant, we have proposed an augmented functionality that prevents the Standstill Situation.

Due to randomness, the PSR OMA automaton might be able to recover from Standstill Situations. However, as explained above, it is a complicating factor that adversely affects the performance of the machine. Therefore, we seek a solution that prevents the Standstill Situation. In our PSR variant, we have therefore proposed a renewed Penalty functionality upon a certain distribution of objects. In the new Penalty operation, we utilize the concept of *Trust*. Although we present the finer details about the concept below, the concept, in itself, can be related to human behavior. We can let the *uncertain* objects infer how they should behave based on the obtained knowledge of *certain* objects. As humans, we might learn from more experienced people, and in our new Penalty functionality that we introduce in the PSR OMA, we utilize the same principle. In order to implement this concept into the PSR OMA, we need to change the inter and intra-state transitions. In what follows, the explanations will be based on the EOMA version of the PSR (the PSR-EOMA).

The first major difference between the existing OMA algorithms and the PSR OMA is the initialization of objects. In the PSR OMA, the abstract objects are initialized in a randomized fashion into the actions (partitions) of the PSR-OMA in accordance with the group sizes that have been established in advance. Similar to the existing OMA algorithms, we distribute objects into the K boundary states in our automaton. But, unlike the existing OMA algorithms, we can have an unequal number of objects in each of these boundary states. The boundary states are the integers given by kS, where $k \in \{1, 2, ..., K\}$, and they will be distributed according to the group sizes given by ρ_k, where $k \in \{1, 2, ..., K\}$. We have O actions in total, K partitions (actions), and each object's state is denoted by ϕ_i, where $i \in \{1, 2, ..., O\}$. We refer to the partitions as ϱ_k for partition k, where $k \in \{1, 2, ..., K\}$. For ease of explanation, we utilize the parameter B_k as a reference to the boundary state of partition k. It is important that we ensure that we have an action (partition) in the automaton that represents each group size, but which of the action numbers it is, i.e., ϱ_k does not matter. We describe the initialization of objects for the PSR-EOMA variant[3] in Algorithm 7 [75]. The reader should note that object initialization is independent of the group sizes, i.e., the process is similar for an EPP and a NEPP.

[3]Because the EOMA version can be easily changed to the Vanilla OMA version, we omit to explain this method in detail here. The reader is referred to Chapter 2 for details about the Vanilla OMA method.

Algorithm 7 PSR Initialization for EOMA

Input:

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

Output:

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

1: **for** all partitions k, where $k \in \{1, 2, ..., K\}$ **do**
2: $B_k = kS$ ▷ The boundary state of ϱ_k
3: $temp = \rho_k$ randomly selected objects from $\mathcal{O}$
4: **for** all objects x in $temp$ **do**
5: $\phi_x = B_k$ ▷ Place object in boundary state of ϱ_k
6: **end for**
7: **end for**

The second major difference is that the PSR OMA algorithm introduces a novel Penalty scheme to address the Standstill Situation, which distinguishes it from both the GCD OMA variants and other existing OMA algorithms. The rationale behind the new Penalty scheme is that when a boundary object (i.e., an uncertain object) is queried together with an object in an innermost state (i.e., an certain object), we utilize the knowledge of the certain object to infer the behavior of the uncertain object. More specifically, when we have one object in a boundary state that is queried together with an object in an innermost state, the boundary object is moved directly to the state of the innermost object, without a replacement if that swap is legal in terms of the partition sizes. If more than one object is required for a legal swap to be done, we will move more objects to the group of the innermost object, and *adaptively* change the partition sizes while the automaton is *in operation*. In the event that relocating the said object does not directly satisfy the stipulated size criteria, an assessment is conducted to determine the presence of other objects within the boundary or state adjacent to the boundary. In the event that the size requirements are met by relocating any or all of the aforementioned entities to the innermost object's partition, the boundary object and its accompanying objects will be transferred to the boundary state of said partition. The concept is visualized on the right side in Figure 4.2, where o_4 is moved to ϱ_3 without a replacement being made.

We present the proposed PSR-EOMA's Penalty operation in Algorithm 8 together with references to Algorithm 9 [75]. The rest of the PSR-EOMA operation is similar to the existing EOMA (the EOMA Reward and EOMA overall operation). We need to emphasize that when moving a single object fulfills the size requirements, we move that object to the same state as the innermost object. When we need to move more than a single object, we are more uncertain that we are moving the correct ones. Therefore, when moving more than a single object, we move them

to the boundary state of the innermost object's action. This behavior can be tuned according to the viewers preferences. If we encounter the scenario when we have all-boundary objects that can be moved legally, we could say that we would anyway move them to the innermost state, because all of them are uncertain. However, in the explanations of the algorithms, we have kept the descriptions to the simplest version, where we distinguish between a single object swap, and multiple object swaps. If no legal swaps that fulfills the size requirement for all groups can be done, we check the rest of the Penalty statements in the EOMA Penalty Process.

Algorithm 8 PSR-EOMA Process Penalty

Input:

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

Output:

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

if ϕ_i mod $S \neq 0$ and ϕ_j mod $S \neq 0$ **then** ▷ Neither are in boundary states
    $\phi_i = \phi_i + 1$
    $\phi_j = \phi_j + 1$
else if ϕ_i mod $S = 1$ and ϕ_j mod $S = 0$ **then** ▷ o_i is in innermost state
    PSR Process for Standstill Situation (Algorithm 9)
else if ϕ_i mod $S = 0$ and ϕ_j mod $S = 1$ **then** ▷ o_j is in innermost state
    PSR Process for Standstill Situation (Algorithm 9)
else if ϕ_i mod $S \neq 0$ and ϕ_j mod $S = 0$ **then** ▷ o_j is in boundary state
    $\phi_i = \phi_i + 1$
    $temp = \phi_j$ ▷ Store the state of o_j
    l = index of an *unaccessed* object in group of o_i closest to the boundary
    $\phi_j = \phi_i$
    $\phi_l = temp$
else if ϕ_i mod $S = 0$ and ϕ_j mod $S \neq 0$ **then** ▷ o_i is in boundary state
    $\phi_j = \phi_j + 1$
    $temp = \phi_i$ ▷ Store the state of o_i
    l = index of an *unaccessed* object in group of o_j closest to the boundary
    $\phi_i = \phi_j$
    $\phi_l = temp$
else ▷ Both are in boundary states
    $temp = \phi_i$ or ϕ_j ▷ Store the state of *Moving* Object, o_i or o_j
    $\phi_i = \phi_j$ or $\phi_j = \phi_i$ ▷ Put *Moving* Object and *Staying* Object together
    o_l =*unaccessed* object in group of *Staying* Object closest to boundary
    $\phi_l = temp$ ▷ Move o_l to the old state of *Moving* Object
end if

Algorithm 9 PSR Process for Standstill Situation

Input:

- The states of all objects ϕ_l, where $l \in \{1, 2, ..., O\}$, and the query $Q = \{o_i, o_j\}$.
- ρ_k for all $k \in \{1, 2, ..., K\}$, and the boundary states, B_k of all $k \in \{1, 2, ..., K\}$.

Output:

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

For ease of explanation, we suppose o_i is in the innermost state of ϱ_i and o_j is in the boundary state of ϱ_j.

if moving o_j to ϱ_i will let our system keep the specified sizes **then**
    $\phi_j = \phi_i$ ▷ Move o_j to ϱ_i
else ▷ If more than one object is required to fulfill all ρ_k
    for all objects o_x in $\varrho_j \setminus o_j$ **do** ▷ All objects in ϱ_j except o_j
        if $\phi_x = \phi_j$ or $\phi_x = \phi_j - 1$ **then** ▷ If o_x is in (or nearest to) the boundary state
            $I \leftarrow o_x$ ▷ I is the set of possible objects to move
        end if
    end for
    if $|\varrho_i| > |\varrho_j|$ **then** ▷ There are more objects in ϱ_i than in ϱ_j
        $\nu = |\varrho_i| - |\varrho_j|$ ▷ $|\varrho_i|$ is the number of objects in ϱ_i
    else if $|\varrho_i| < |\varrho_j|$ **then** ▷ There are more objects in ϱ_j than in ϱ_i
        $\nu = |\varrho_j| - |\varrho_i|$
    else ▷ This means $|\varrho_i| = |\varrho_j|$
        Continue Process Penalty ▷ Continue with the remaining statements in Alg. 8
    end if
    if $|I| + 1 \geq \nu$ **then** ▷ The number of objects in I are bigger than (or equal to) ν
        Randomly select $\nu - 1$ objects from I and put them in a new set J.
        if $|\varrho_i| + \nu$ and $|\varrho_j| - \nu$ fulfills all ρ_k **then** ▷ If the size requirement is fulfilled
            $\phi_j = B_i$ ▷ Move o_j to boundary of ϱ_i
            for all objects o_z in J **do**
                $\phi_z = B_i$ ▷ Move objects in J to boundary state of ϱ_i
            end for
        end if
    else ▷ It was not possible to make a legal swapping of objects
        Continue Process Penalty ▷ Continue with the remaining statements in Alg. 8
    end if
end if

The PSR OMA is a new variant of the OMA algorithm that can solve partitioning problems with arbitrary partition sizes without requiring a common non-unity GCD between the group sizes. This is achieved by adaptively swapping the group sizes within its operation. However, the PSR variant still requires pre-specified cardinalities and can lead to a Standstill Situation where a smaller partition is trapped inside a larger partition, preventing objects from being grouped together. To address this issue, we have proposed a renewed Penalty functionality that utilizes the concept

of trust, where uncertain objects infer how to behave based on the knowledge of certain objects. This concept is implemented by changing the inter and intra-state transitions of the automaton.

4.2 Chapter Summary

In this chapter, we have introduced two new variants to the OMA paradigm, namely the GCD OMA and the PSR OMA. They can be utilized by invoking all the existing OMA algorithms, and allow them to be able to handle both EPPs and NEPPs. The existing OMA algorithms can only handle partitioning problems where the groups have equal sizes, but the proposed variants do not have this limitation. The GCD OMA requires a non-unity common GCD between the partition sizes, while the PSR OMA does not have a size relation requirement to the group sizes of the problem. The GCD OMA extends actions onto a larger state space. On the other hand, the PSR OMA has a unique Penalty operation upon a particular distribution of objects, a new initialization of objects, and other inter and intra-state policies allowing it to solve NEPPs of arbitrary sizes. These innovations are, truly, remarkable to their field, and the OMA algorithms can now solve a larger selection of grouping problems than they were able to achieve prior to their invention.

Chapter 5

Novel VSSA Solutions

As emphasized in the previous chapters, the VSSA type of LA differ significantly from their fixed-structure, FSSA, counterparts. In the preceding chapter, we introduced our innovative contributions to the field of FSSA research. This chapter will focus on our pioneering work in the VSSA domain. Similar to our FSSA contributions, our advancements in the VSSA domain also encompass entirely new learning schemes. However, we have delved even deeper into the underlying algorithms by conducting rigorous mathematical analyses. Specifically, we have established the ϵ-optimality of the HDPA, and for the Action Distribution Enhancing (ADE) HDPA, we elucidated the fundamental mathematical principles behind our proposed enhancement of the HDPA implementation. More specifically, our contributions include our novel HDPA and ADE HDPA, respectively. These innovations represent the state of the art within the field of LA.

The VSSA machines are distinguished by their capacity to encapsulate memory in the form of a probability vector, allowing them to monitor interactions with the Environment and record those interactions. However, as explained earlier, the problems become more complicated as the number of actions, R, increases. The learning mechanism in continuous VSSA LA operates in a multiplicative manner. In most cases, the appropriate action selection probability is initialized with the value $\frac{1}{R}$ as the starting probability for each action. Consequently, the likelihood that a certain action would be picked as the starting point reduces as R increases. Therefore, the LA needs a smaller learning parameter to ensure genuine learning if it is to be achieved by the corresponding updating methods. Thus, the LA should not just instantly boost the probability of one of the actions to the point where it becomes the only action that the machine chooses, without exploring the entire action space. These two phenomena can result in inaccurate and slow convergence as R increases. In [2], the authors proposed the HCPA to resolve the issue of "Environments with a large number of actions". The HCPA represented the state-of-the-art algorithm in VSSA before our innovations [2].

Even though it is a very fast machine, the HCPA suffers from a relatively slow convergence as the accuracy requirement becomes high. Thus, when any of the action's probability in the action selection probability vector approaches unity, the change in its probability becomes correspondingly smaller. For example, let us

assume that the action probability of one of the actions, say α_1, in a two-action Environment, is currently $p_1 = 0.999$. Correspondingly, the action selection probability of the second action, say α_2, is $p_2 = 0.001$. Furthermore, let us assume that we have a continuous learning scheme with a learning parameter, $\lambda = 0.001$, and that α_1 is selected and receives a Reward. The action selection probability of α_2 is then updated according to the following formula: $p_2 = (1 - \lambda)p_2$, resulting in $p_2 = 0.000999$. Further, the action probability of α_1 is updated according to the formula: $p_1 = 1 - p_2$, i.e., $p_1 = 0.999001$. Consequently, the change in the action probability of α_1 is 0.000001. In comparison, if α_1 had an action selection probability of $p_1 = 0.57$ initially (and $p_2 = 0.43$), the corresponding change in action probability would be 0.00043.

The behavior demonstrated above, results in *sluggish* convergence as the LA gets closer to achieving convergence. Consequently, even if the HCPA can tackle a *large number of actions*, a more effective behavior for high accuracy requirements would make the algorithm better suited for the future. Another aspect related to the algorithm's effectiveness is that the HCPA organizes its LA in a tree structure, with its second bottom LAs serving as the automatons responsible for the actual leaf actions that interact with the Environment. The leaf actions are selected through various pathways through the tree, and one would question if the *ordering* of the actions at the leaf level could impact the overall machine's performance. Therefore, we were motivated to see if we could advance the field of VSSA further by proposing new learning schemes that attain convergence faster while accurately converging to the optimal action.

Our research has made two novel advancements in VSSA, with the introduction of our proposed HDPA variant and the ADE HDPA, respectively. The HDPA concept aims to enhance the performance of VSSA by organizing a set of DPA instances in a hierarchical tree structure. This approach simultaneously discretizes the probability space, structures the LA instances, and incorporates the Estimator concept. The HDPA variant mitigates the sluggish LA performance, especially for high accuracy requirements. The second novel scheme we propose is the ADE HDPA, where we suggest a new methodology to structure the actions at the leaf level of the tree. These developments reflect the most recent and advanced state-of-the-art algorithms in the VSSA paradigm, and they will be presented briefly below.

The comprehensive details of our VSSA contributions are available in the supporting papers incorporated later in this dissertation. This chapter aims to deliver a high-level overview of the advancements made in the VSSA domain. To avoid redundancy and ensure that the proposed concepts are adequately contextualized, we present the VSSA contributions more concisely than the preceding chapter about the novel FSSA contributions.

5.1 The HDPA

Although the HCPA represents a suitable solution (and it was the best available LA) when we had many actions, it suffers from slow convergence, especially in the latter phase of learning, as exemplified above. As the machine approaches the convergence criterion, the ineffective updating scheme becomes especially noticeable when we have high accuracy requirements, e.g., a convergence criterion above, for example, 0.99. When we have high accuracy requirements, the LA has to reach a higher action selection probability threshold. Because the change in probability becomes correspondingly less as the action selection probability approaches unity, it is clear that the sluggish convergence behavior becomes more prominent for higher accuracy requirements. In other words, the changes in the probability vector become less as the number of iterations and interactions with the Environment increases, resulting in more iterations being required before convergence is achieved. A more effective learning scheme would increase the applicability of these types of ML solutions.

In order to enhance convergence speed in situations where we have environments with a large number of actions, and we have high requirements for the accuracy of the converged solution, we have proposed the HDPA variant. The benefit of the HDPA is that it does not have the same impediment as the HCPA. Thus, the effective learning rate remains strong as the learning progresses. We achieve this behavior by piggybacking the phenomenon of discretization into the machine mechanism and incorporating the earlier-explained innovations presented in Chapter 3, namely of discretization, the Estimator phenomenon, and structure.

The HDPA represents a new algorithm within the VSSA domain. For the HDPA, we establish a hierarchical tree arrangement for a collection of DPA instances, wherein each DPA instance is associated with a distinct set of actions that correspond to the potential paths throughout the tree structure. The action probabilities of the corresponding LAs are maintained via vectors that are subject to discretized updates contingent upon whether a given action incurs a Reward or Penalty. At the lowest tier of the hierarchical structure, the actions are situated in direct interaction with the Environment. The reward estimates of all the actions at the leaf level of the tree are maintained in vectors that are distinct from the action selection probability vector maintained in each LA. As explained in the respective papers attached, action estimates are, in reality, only needed for the actions at the leaf level[1]. As per the Pursuit paradigm, the HDPA pursues the currently *best* estimated action in each iteration. Consequently, the action that possesses the highest estimated probability of getting a Reward is granted an increase when a Reward is received, irrespective of which action was responsible for triggering the Reward. Estimator algorithms have been demonstrated to achieve a far superior convergence faster than their predecessors [2].

[1]However, as explained in subsequent papers, the estimation in every LA is needed to prove its ϵ-optimality, and define its behavior in a greater and more precise detail.

5.1.1 Motivation for this Study

In today's society, the amount of data stored constantly increases, and the Internet requires significant energy resources. Similarly, our machine learning algorithms add to our carbon footprint [80, 81]. Therefore, it is essential to prioritize sustainability by persistently striving to reduce the resources required to execute the various algorithms. In this context, improving the schemes' efficiency is crucial to the future of these algorithms. Algorithmic efficiency is not only essential from a sustainability standpoint, but it is also a fundamental consideration in the algorithm's applicability to real-life problems. Inefficient algorithms reduce the likelihood of their utility in contexts where the systems are highly stochastic, and which might change rapidly. In these cases, the algorithms that we use need to be efficient.

One of the most important aspects in the Literature on LA, has always involved improving the speed and accuracy of the algorithms. Finding new ideas and principles to change the machines' inner workings has been necessary to achieve better and more effective learning schemes. Earlier, we discussed all of the significant developments that have advanced the field until today, i.e., the introduction of learning and memory as maintained in a probability vector, the discretization of the probability space, the Pursuit concept, and the concept of structure. The combinations of these innovations have led to even further improvements, like in the case of larger action spaces, where the ϵ-optimal HCPA is currently the record holder for such Environments. However, as already highlighted, we wanted to advance the algorithms even further, impacting the field of VSSA in itself. Our proposed HDPA represents our first novel contribution to the area of VSSA, representing a new record-holder in both speed and accuracy.

5.1.2 Principles for the HDPA

In this section, we will explain the principles for the HDPA in a high level[2]. In the HDPA, we combine the concepts of VSSA, discretization, the Pursuit concept, and structure. More specifically, DPA instances are organized in a hierarchical tree structure, where each automaton has a set of actions that lead to a child automaton. Let us define the number of levels in the tree structure[3] as L. The automata at tree depth $L-1$ are responsible for the actual actions interacting with the Environment. Thus, the *children* of these automatons are not new automatons, but the leaf actions themselves. We visualize an example structure of an HDPA hierarchical tree in Figure 5.1, which shows 8 actions structured in a tree with $L = 3$ and $R-1$ automata. For R actions, we need $R-1$ two-action DPA instances. The relation between the number of actions and the required tree-depth is $R = 2^L$.

[2]The details of the HDPA algorithm are presented in its entirety in the subsequent papers, included in this dissertation.

[3]In the subsequent papers addressing the HDPA, K is used as a notation for the tree depth. For the reader to not be confused with the symbol K, used as the number of partitions in the OMA paradigm in the earlier chapters, we have used the symbol L to refer to the tree depth, in this chapter.

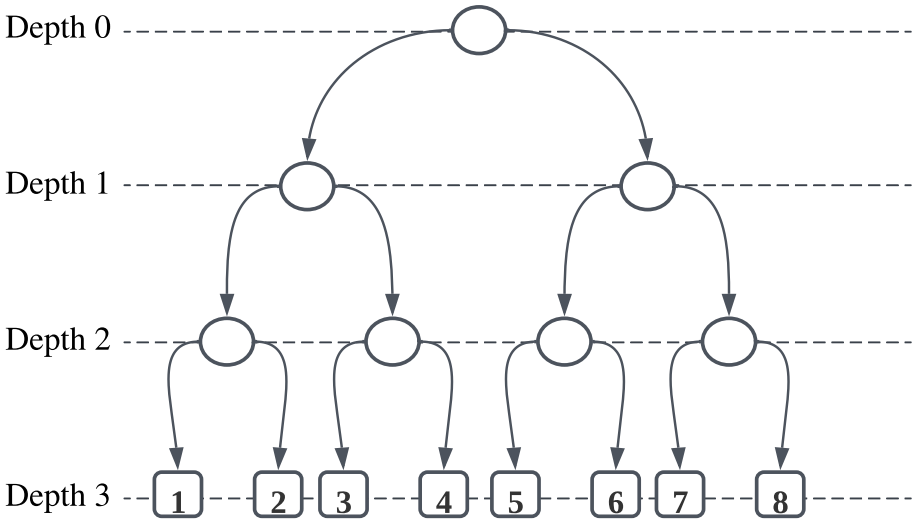


Figure 5.1: An example of an HDPA structure for eight actions.

The LA instances in the tree can be of any type, with each of them having an arbitrary number of actions. In this dissertation, and the subsequent attached papers, we have utilized two-action DP_{R-I} instances in the tree structure. The reader should note that the instances could have possessed more than two actions, but we should remember that the overall reason for the hierarchy is to reduce the dimension of the action probability vector. If the number of actions is not a multiple of two, we use a multiple of two with a number of actions above the number we require, and set some of the actions to be inactive, with a reward probability of zero.

Let us further define the notations for the hierarchy as follows[4]:

- **The depth index of the tree**: For a tree whose depth is L, we employ l to index the tree depth, where $l \in \{0, 1, \ldots, L\}$.
- **The LA in the structure**: We denote the individual LA by $\mathcal{A}_{\{l,j\}}$, where l refers to its depth and j represents the index of the LA within the depth l. Consequently, the LA $j \in \{1, ..., 2^l\}$ at depth l, is denoted as $\mathcal{A}_{\{l,j\}}$, where the depth l, $l \in \{0, ..., L-1\}$. Observe that there are no LA at depth L, because they are the actions themselves.
- **The LA in depths** 0 **to** $L-1$**, i.e.,** $0 \leq l < L-1$:
  - Each of the LA, $\mathcal{A}_{\{l,j\}}$, has two actions, which we denote as $\alpha_{\{l+1,2j-1\}}$ and $\alpha_{\{l+1,2j\}}$, respectively.
  - When the action $\alpha_{\{l+1,2j-1\}}$ is selected, i.e., the left branch of the current LA, the LA $\mathcal{A}_{\{l+1,2j-1\}}$, at the next depth of the tree is activated.
  - When the action $\alpha_{\{l+1,2j\}}$ is selected, i.e., the right branch of the current LA, the LA $\mathcal{A}_{\{l+1,2j\}}$, at the next depth of the tree is activated.

[4]This notation is similar to the one established in [2], and the explanation given in subsequent attached papers to this dissertation.

– We refer to $\mathcal{A}_{\{l+1,2j-1\}}$ and $\mathcal{A}_{\{l+1,2j\}}$ as the *Left Child* and the *Right Child* of the Parent ($\mathcal{A}_{\{l,j\}}$), respectively.

- **The LA in depth $L-1$ (i.e., $l = L-1$)**: The LA at depth $L-1$ are responsible for choosing the activated action at the leaf level, i.e., the action that is *chosen* and which interacts with the Environment.

  – Each LA at depth $L-1$ has two possible actions, which we refer to as the left and right child. Formally, these are denoted as $\alpha_{\{L,2j-1\}}$ and $\alpha_{\{L,2j\}}$, respectively.

  – Inside depth $L-1$, the LA are responsible for 2^L actions, and the actions at the leaf level are referred to as $\alpha_{\{L,j\}}$ where $j \in \{1, ..., 2^L\}$.

  – If we consider the chosen action at the leaf level to be: $\alpha_{\{L,j\}}$, we can determine its parent from the following relation: $\mathcal{A}_{\{L-1,\lceil j/2 \rceil\}}$.

- **The actions at depth L ($l = L$)**: At depth L, we have the actions that directly interact with the Environment.

The above notations and concepts are crucial for understanding the operation of the HDPA. The principle of the HDPA is that an action is activated at the leaf level from a path from the top node to a node at the $L-1$ level through the tree. The additional details of this operation will be presented presently.

5.1.3 Summary of the Overall HDPA Algorithm

In this section, we delve even deeper into the operation of the HDPA. As we can observe from Figure 5.1, each node, except the ones at depth $L-1$, is the parent of two children. The procedure of the HDPA starts with the LA at the top of the tree structure, referred to as the *root node*. The root node, $\mathcal{A}_{\{0,1\}}$, has an action selection probability vector, p_{01}, maintaining its probability for choosing either its Left or Right child, respectively. By a random uniform sampling of this probability vector, the LA at the next level in the tree is activated. A similar process is continued until level $L-1$ is reached. The reader should note that the action selection probabilities are followed for choosing a leaf action, and that the reward estimates are not a part of the action selection process. Instead, the reward estimates are only a part of the updating and learning mechanism of the machine.

When the LA at depth $L-1$ has chosen an action at the leaf level by a uniform random sampling of its action selection probability vector, the action interacts with the Environment. The Environment will respond, either with a Reward or a Penalty[5]. Because we consider DP_{R-I} instances in this dissertation, a Penalty does not result in any updating of the action selection probability vector. However, the reward estimates vector is updated both if a Reward or Penalty is received. More

[5]As we have mentioned earlier, the feedback from the Environment does not necessarily come from a binary set of responses. However, in this dissertation, we only consider the scenario where the Environment responds with either a Reward or a Penalty.

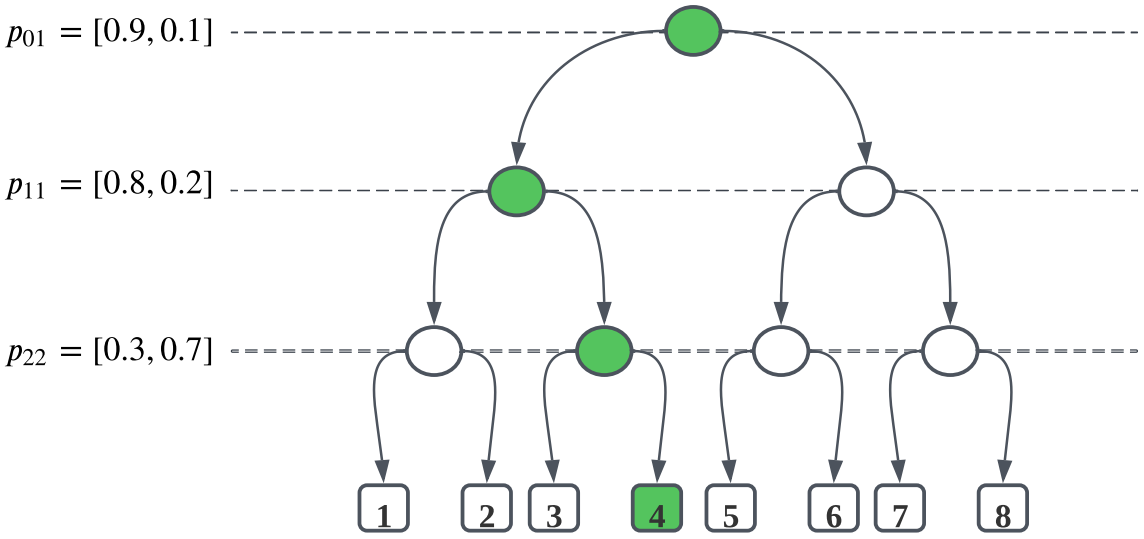
specifically, we maintain reward estimates for all the leaf actions. For example, the reward estimate of α_1 might be 0.9 if that action has been selected 100 times and received a Reward 90 out of the 100 interactions with the Environment.

We base the updating mechanism of the HDPA on feedback from the Environment. However, in the HDPA, we follow the Pursuit concept. Thus, if α_1 is the action selected and the one that received a Reward from the Environment, but α_2 currently has the highest reward estimate, we will reward the path to that action instead of the action that was selected. In this way, we pursue the currently best-estimated action in every iteration. More specifically, when a leaf action is selected and receives a Reward, we will reward the backward (or rather, upward) path leading to the action that currently has the best reward estimate. This process is continued until convergence is achieved. Convergence can be defined in different ways for the HDPA. We can either say that the HDPA converges once *any* of the action selection probabilities is above a specified threshold, or we can say that the HDPA only converges once a complete path throughout the tree in terms of the action selection probabilities, *have converged*, i.e., all the LA on the path to a particular action has an action probability above a certain threshold.

Let us consider a more detailed example of the HDPA principle upon a Reward, where we have eight actions as depicted in Figure 5.2. Furthermore, we define our learning parameter as $\Delta = 0.001$, and we let $\hat{D}$ denote our reward estimate vector. The reader should remember that the reward estimate vector keeps track of the number of times an action has been selected and has received a Reward from the Environment. Let us consider that $\alpha_{3,4}$ was selected. From the reward estimate vector in Figure 5.2, we observe that the selected action happens to be the action with the highest estimated reward probability. The activated LA for $\alpha_{3,4}$ to be chosen is colored green. The current action selection probabilities of the green automatons on the path are listed on the left side of the figure. To select an action, we follow the path down the tree by sampling the action probabilities in the LA along the path. For example, when $\mathcal{A}_{\{1,1\}}$ has an action probability vector of $[0.8, 0.2]$, it selects $\alpha_{\{2,2\}}$, with probability 0.2.

In Figure 5.2 referred to above, we saw that $\alpha_{3,4}$ was selected. Let us assume that this action triggered a Reward from the Environment. In such a case, we will update the reward estimate of that action, as highlighted by the blue color in $\hat{D}$ in Figure 5.3. From Figure 5.3, we can observe the changes in action selection probability along the path leading to the action with the highest estimated reward probability given by $\hat{D}$. Whether we update the reward estimate before or after, the path updating can be defined according to the executor's preferences[6]. Consequently, we reward the reverse path from $\alpha_{3,4}$, as demonstrated in Figure 5.3. The updated LA are marked in blue, and their corresponding action probabilities that experience a change are

[6]The *order* of the operation in our HDPA is defined in the subsequent attached papers describing the HDPA.



$\hat{D} = [0.54, 0.76, 0.33, 0.98, 0.59, 0.48, 0.23, 0.35]$

Figure 5.2: The HDPA principle, where an action at the leaf level has been chosen.

also marked in blue. As an example, p_{11} is updated as follows:

$$p_{11}[0] = \max(p_{11}[0] - \Delta, 0), \tag{5.1}$$

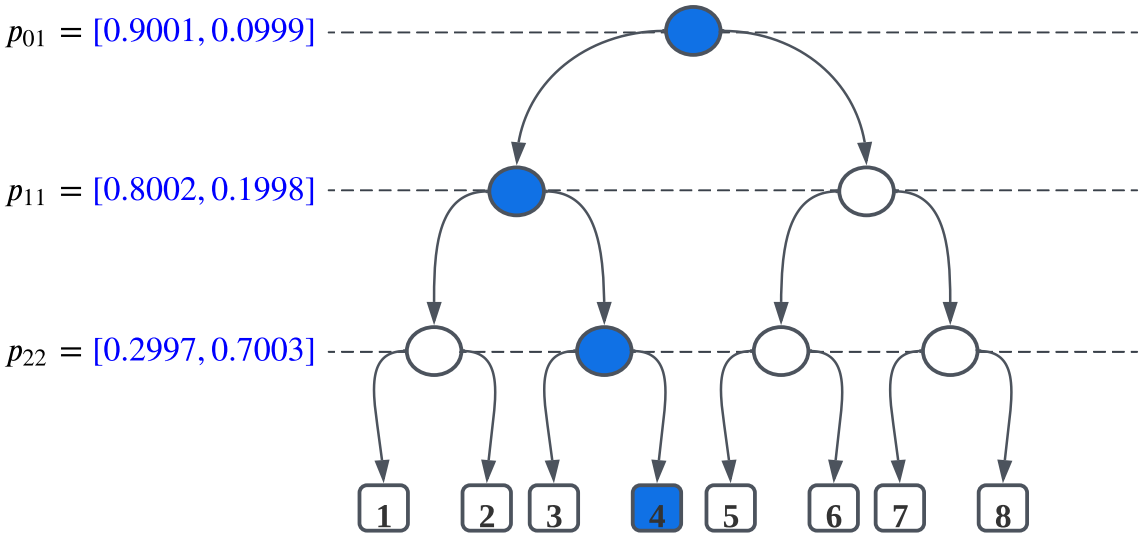
$$p_{11}[1] = 1 - p_{11}[0], \tag{5.2}$$

where $p_{11}[0]$ represents the Left child of $\mathcal{A}_{\{1,1\}}$, and $p_{11}[1]$ represents the probability for choosing the Right child. The reader should note that, for the ease of understanding, we have omitted n in this example. Consequently, for this automaton, we reward the right branch (increase the probability of selecting the Right child).

From the example described above, we understand the overall concept of the HDPA. We want to emphasize that if another action than $\alpha_{3,4}$ had the highest estimated reward probability, we would, as per the Pursuit concept, have rewarded the path leading to that action instead. We have constructed this example so as to make the concept easy to follow.

5.1.4 Overview of the HDPA Results

Extensive simulations were conducted to showcase the effectiveness of the proposed HDPA scheme in various environments featuring a multitude of different numbers of actions, and other configurations. These simulations are detailed in the attached papers of this dissertation, and we will not completely reproduce them here. However, in Table 5.1 and Table 5.2, we have extracted some of the results to compare the HCPA algorithm with our proposed HDPA algorithm. From the tables, we can observe that the HDPA has significantly fewer average required number of iterations before convergence, but that of the HCPA has a smaller standard deviation, in most cases, compared with the HDPA. The reader should note that the "difficulty" associated with the Environments, sometimes referred to as its "hardness", is significantly different for various numbers of actions, e.g., making the 512 actions Environment



$\hat{D} = [0.54, 0.76, 0.33, 0.99, 0.59, 0.48, 0.23, 0.35]$

Figure 5.3: The HDPA principle, where we update the reward estimate and path to the leaf action with the highest estimated reward probability.

easier than the case with 256 actions. The hardness is, e.g., related to how near the sub-optimal and optimal action are, in terms of their reward probabilities in the Simulation Environment.

Number of actions	HCPA	HDPA
128	155,100	97,800
256	1,039,200	633,000
512	95,400	78,800

Table 5.1: A brief synopsis of the simulation results over an average of $1,000$ experiments for the HCPA and HDPA. The table lists the average number of iterations required for different Environments.

Number of actions	HCPA	HDPA
128	10,600	13,300
256	88,700	71,000
512	13,100	15,400

Table 5.2: A brief synopsis of the simulation results over an average of $1,000$ experiments for the HCPA and HDPA. The table lists the standard deviation of the number of iterations required for different Environments.

As previously noted, a limitation of the HCPA is its slow rate of updating action probabilities as the probability of increasing them approaches unity. In contrast to this, it was hypothesized that the HDPA, characterized by a consistent increase in the action selection probability in the updating functionality, would exhibit considerably better convergence compared to the HCPA, which has been confirmed in

the rigorous simulations we conducted. Consequently, our proposed HDPA is significantly more effective regarding the average number of iterations required before convergence for high accuracy requirements (e.g., above 0.99).

5.2 The ADE HDPA

The reader will notice that the ordering of the actions has remained unimportant throughout the introduction and development of VSSA. This is valid because the ranking cannot be determined without resorting to a prior Random Race competition [52] and because, in general, there is no relevance to the ordering of the actions in a standard action selection probability vector. Thus, the order of the actions in the action probability vector is meaningless, and has no impact on the effectiveness of the machine. However, when we have hierarchical structures of LA, as in the case of the HDPA, we have discovered that the ordering matters for the algorithm's effectiveness in terms of the number of iterations required before the algorithm converges. To confirm this hypothesis, we have proposed the ADE HDPA variant, where we consider the ordering of the actions, and this is the avenue which we will discuss in more detail below.

In the HDPA, we have automata that operate hierarchically, where the actions that interact with the Environment are placed at the bottom of the tree. A leaf action is selected based on the action selection process throughout the tree structure, from the root node to the nodes at the "second bottom" level of the tree. Consequently, we hypothesized that the order of the actions at the leaf level could influence the performance of the machine, because if two actions are close in terms of their *real* reward probabilities, it requires more iterations if they are located far apart in the tree structure as opposed to them being placed closer together. In simple terms, the reason for this behavior is because it requires more effort for the LA at each depth of the tree when they have more potentially optimal paths to discriminate between, which happens when, e.g., the least sub-optimal and optimal actions are located far apart. On the other hand, if the LA are concerned about distinguishing between fewer potentially optimal paths (actions), they operate more effectively, which happens when the sub-optimal and optimal actions are located closer together at the leaf level of the tree structure. Therefore, the ADE approach is based on the concept of ordering the actions in an ascending or descending order based on the reward estimates, which ensures an optimal distribution of the actions for an efficient operation for the LA. In Figure 5.6, we have visualized different action distributions at the leaf level of an HDPA machine.

The reader should note that in a real system, we do not have any prior knowledge of the actions' reward probabilities. By distributing the actions into random positions at the leaf level, we might be *lucky*. Thus, the actions might, with a small probability, be ordered in an ascending or descending order naturally. However, it is better to reduce this uncertainty, and this is what we can achieve by implementing the ADE approach. In the simulations we conducted to demonstrate the improve-

ment that can be achieved with the ADE approach, which we summarize in the section below, the proposed algorithm did not necessarily perform better when the actions were already ordered in an ascending or descending order according to their reward probability. The reason for this behavior is that the ADE approach estimates the reward probabilities obtained stochastically and, therefore, does not necessarily represent the actions' real reward probabilities correctly. We do not know the real reward probabilities up-front, and if we did, it would nullify the learning problem, and render it to be void. Therefore, estimating the reward probabilities and then reorganizing the actions at the leaf level according to the ADE approach represents a pioneering and suitable solution to improve the effectiveness of hierarchical LA structures further.

The reader should note that the ADE approach can be implemented into *any* machine with hierarchically structured LA, e.g., the HCPA or the HDPA. However, in the remaining parts of this dissertation, we would focus on the HDPA variant. Because we have already highlighted its benefit over the HCPA, and in the interest of simplicity, we thus proceed with the discussion by considering the case of the HDPA.

5.2.1 Motivation for this Study

As Section 5.1 explains, it is extremely pertinent for the designer to improve the algorithms' efficiency. We have, thus, similar motivational arguments for proposing the ADE approach as the ones we described earlier. Indeed, the algorithm's efficiency is an essential topic in the interest of sustainability and the applicability of VSSA algorithms to practical problems. However, to further motivate the development of our new ADE paradigm, let us consider an example where we have four actions, denoted as α_1, α_2, α_3, and α_4, respectively[7]. Let us assume that we have estimated the reward probabilities of these actions, and that their reward *estimates* are $\hat{D} = \{\hat{d}_1 = 0.9, \hat{d}_2 = 0.3, \hat{d}_3 = 0.5, \hat{d}_4 = 0.8\}$, respectively. Let us assume that we estimated the reward probabilities by testing each action with the Environment, a few, say ten times. Thus, the reward estimate $\hat{d}_1 = 0.9$ means that α_1 received a Reward nine times out of the ten times were we tested that action.

We visualize the problem discussed above in Figure 5.4. We can place the actions in one of the 4! possible positions. In the figure, we have placed the actions randomly at the leaf level. In Figure 5.5, we visualize a configuration where the actions are placed in a descending order[8]. The LA, in different depths, deals with different complexities. The root node is the most important node for the operation of the LA, and it influences all the LA below it. In this example, the LA at the level below the root node governs no other LA, except its leaf actions.

In Figure 5.4, we observe that the node governing α_1 and α_2 has to deal with

[7] A similar example is highlighted in the attached papers in this dissertation about the ADE approach.

[8] The reader should note that a descending and an ascending order are merely mirrored reflections of one another.

actions whose reward estimates are $\hat{d}_1 = 0.9$ and $\hat{d}_2 = 0.3$, respectively. In Figure 5.5, the actions are ordered according to their reward estimates, and the same node has to deal with distinguishing between actions with the reward estimates $\hat{d}_1 = 0.9$ and $\hat{d}_4 = 0.8$, respectively. If we trickle "up" the maximum of the reward estimates to the root node, the root node has to deal with distinguishing between the reward estimates 0.9 and 0.8 in Figure 5.4, and between the reward estimates 0.9 and 0.5 in Figure 5.5. This implies that the root automaton, which must resolve the most challenging issue, has a more difficult task in the configuration of Figure 5.4, when compared with the corresponding configuration of Figure 5.5. Understandably, it is easier for the root node to distinguish between actions with a higher difference in reward probabilities. Consequently, we want to move the most discriminating problem closest to the leaf level of the tree, which can be achieved by ordering the actions in a ascending/descending order according to their reward estimates[9]. Thus, the discriminating problem becomes harder with the depth of the tree. The explained example of how the action distribution at the leaf level influences the operation highlights the reason and motivation for considering the ordering of the actions.

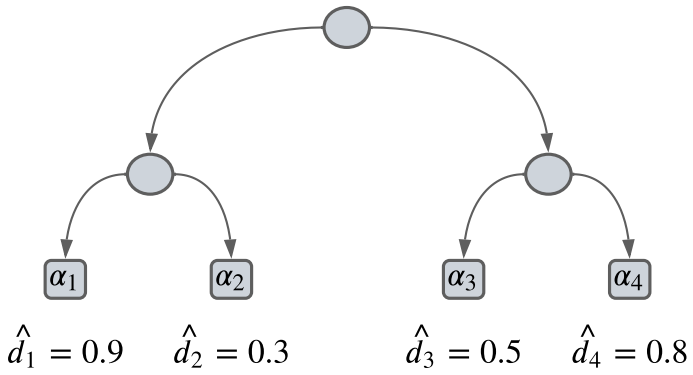


Figure 5.4: Example of an action distribution with the actions in a random order.

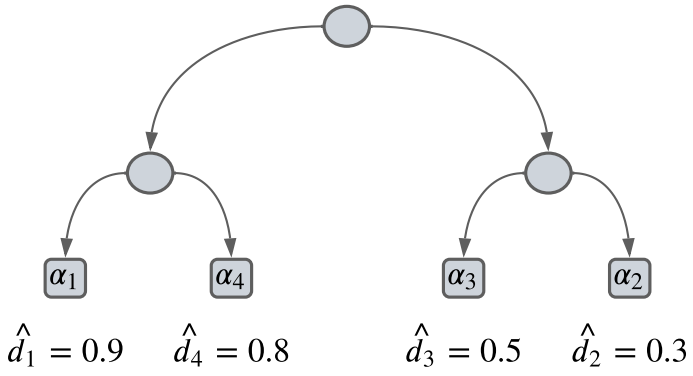


Figure 5.5: Example of an action distribution with the actions in descending order based on their reward estimates.

[9]The formal mathematical analysis why this is true, is presented in the attached journal paper in this dissertation, which deals with the ADE.

5.2.2 Principles for the ADE HDPA

The principle of the ADE approach is to distribute the actions at the leaf level of the tree in a more optimal manner, to achieve faster convergence in hierarchical LA structures. In order to optimally distribute the actions at the leaf level, we need to order them in an ascending or descending manner. In this way, we will ensure that the most discriminating problem lies with the root node, and the problem of distinguishing between optimal and next sub-optimal actions in, e.g., a binary tree, becomes more complicated as we move down the hierarchical depths of the tree.

Figure 5.6 shows examples of different action configurations at the leaf level. If we focus on the sub-optimal and optimal actions in the figure, the configuration of Figure 5.6a) represents the *most effective* configuration, while that of Figure 5.6b) represents the *least effective* configuration at the leaf level. The two other configurations (the Figure 5.6c) and Figure 5.6d) configurations) represents distributions in-between the hardness level of the most effective and least effective configurations. The reader should note that the other actions' probabilities also influence the difficulty of the Environment. In general, it takes fewer iterations for an LA to solve a problem where the real reward probability of the optimal action is significantly higher than the other actions' reward probabilities [4].

We normally have sparse to non-existent *a priori* information about the different actions' fitness for a practical, real-life problem. Thus, their reward probabilities are unknown. As a result, we must gather data so that we have information based on which we can asses the distribution at the leaf level. Therefore, the principle of the ADE requires two processes. More specifically, with minimized sampling, we need to first gather information about the actions' reward probabilities. Thus, we need to initially estimate the reward probabilities, which we refer to as the *Estimation Phase* of the ADE approach. The second phase involves the actual ordering of the actions, referred to as the *Reallocation Process* of the ADE approach. By implementing these two steps, we can distribute the actions in an improved order in such a manner that the LA can operate more effectively.

5.2.3 Summary of the Overall ADE Algorithm

We will not describe the ADE HDPA algorithm in detail here because it is presented, in its entirety, in the papers included in this dissertation. However, we will here discuss only the steps required for the algorithm, in a general manner. As mentioned above, the ADE approach consists of two processes, the *Estimation phase* and the *Reallocation phase*. We will explain these two steps in more detail below. As we can observe from Figure 5.7, within the LA, we have the Estimation Phase and the Reallocation processes, and these steps need to be continued by the regular operation of the machine. The hierarchical LA method can, in and of itself, be of *any* type, but in this figure, we have suggested the HCPA and the HDPA algorithms as examples.

Let us first consider the first part of the ADE approach, namely the Estimation Phase. With the HDPA and other LA that utilize the concept of Estimation, we

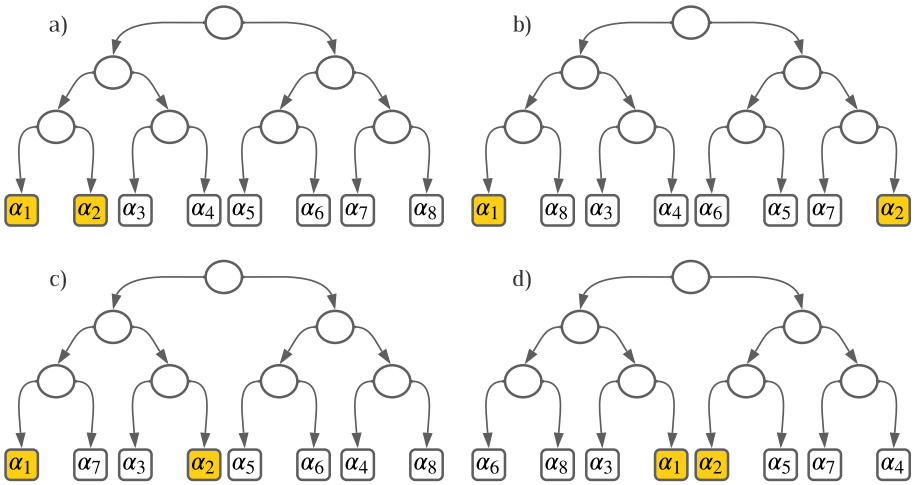


Figure 5.6: Examples of different action distributions at the leaf level, where the colored actions are the sub-optimal and optimal actions, respectively.

already include a vector (or vectors) for maintaining the reward estimates of the machine. Typically, these estimates (and the action probabilities themselves) are initialized as 0.5 [2]. With the ADE approach, we propose to initiate a standalone Estimation Phase before we proceed with the other already-mentioned steps. More specifically, in this step, we include θ initial iterations per action, where they are tested with the Environment. We record the reward estimates, and based on these, we proceed with the Reallocation Process. The reader should note that the reward estimates should be kept, but the vector needs to be reordered according to the new positions of the actions after the Reallocation Process. However, in the Estimation Phase, no learning is conducted. We emphasize that the rough (or crude) estimates might require several iterations, which are, indeed, not utilized for learning[10].

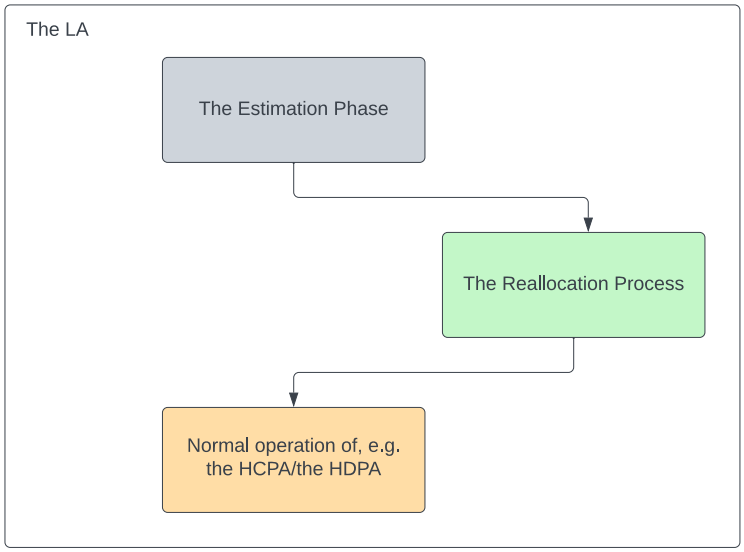


Figure 5.7: A visualization of the processes within the ADE approach.

[10]In our simulations, detailed in the attached papers, we demonstrated that the benefit of the ADE typically far exceeds the number of iterations required for the Estimation Phase.

The second part of the ADE approach is the Reallocation Process. In this step, we distribute the actions in an improved manner based on the reward estimates established in the Estimation Phase. As proven in the attached journal ADE paper, enhanced organization can be achieved by ordering the actions at the leaf level in an ascending or descending order. Consequently, the tree's leaves are organized in an ascending/descending sequence using the estimated reward probabilities. Thus, the actions are given a new location at the leaf level, which, as mentioned, is referred to as the Reallocation Process. We emphasize that due to the stochastic nature of the problems in the LA domain, the estimation of the reward probabilities may not always yield a perfect representation of the actual reward probabilities. Thus, the Reallocation Process distributes the actions in an improved manner. After the Reallocation Process, the algorithm continues its regular operation, and the reward estimates are maintained to ensure that this information is preserved and not lost.

5.2.4 Overview of the ADE HDPA Results

To demonstrate the validity of our hypothesis, we conducted experiments with different configurations to demonstrate the effectiveness and improvement of considering the ordering of the actions in a hierarchical machine with the ADE approach. We ran simulations with the HDPA variant, and in all of the configurations listed here, the ADE HDPA performed better than the HDPA variant. Table 5.3 and Table 5.4 summarize the results of some of these experiments. These results are extracted from the papers attached to this dissertation, and the details can be found there. The results listed in the tables are approximated and simplified to demonstrate the improved efficiency of the ADE HDPA compared with the HDPA, in a general manner.

As we can observe from Table 5.3, we see that in all of the listed Environments, the ADE HDPA performed better than the plain HDPA. We can observe that the benefit of the ADE HDPA increased with the number of actions. For example, for 64 actions, the HDPA required on average approximately $91,700$ iterations before convergence, while the ADE HDPA required approximately $75,100$ iterations, corresponding to approximately 18% improvement. Also, for the 8-action case, the ADE HDPA yielded a 4% improvement to the average number of iterations required before convergence, compared with the HDPA. From Table 5.4, we can clearly observe that the standard deviation achieved with the ADE HDPA is less than for the HDPA. The benefit of the ADE approach for the standard deviation also increased as the number of actions increased. From our simulations, we observed that the ADE implementations reduced the instability from one simulation to another with different leaf-level action distributions. Thus, the ADE approach performs similarly for different initial action distributions because they are reorganized in the process, where the individualities and influence of the actions' locations at the leaf level of the tree are eliminated.

In the performance evaluation of LA machines, we talk about the number of iterations required before convergence to measure the algorithms' effectiveness. The

reader might ask why, e.g., execution time and memory utilization are not measured instead. This is because the average number of iterations metric remains constant irrespective of the computational capacity of the machine employed for conducting the experiments, the way the algorithm is programmed (which depends on the programmer's competence), and the utilized code or programming language. Consequently, the average number of iterations required before convergence is a transferable and stable comparable metric, especially for research purposes.

Number of actions	HDPA	ADE HDPA
8	7,100	6,800
16	12,700	11,600
34	17,700	16,300
64	91,700	75,100

Table 5.3: Simplified simulation results for the HDPA and ADE HDPA. The table lists the average number of iterations required for different Environments.

Number of actions	HDPA	ADE HDPA
8	220	50
16	510	110
34	850	120
64	10,270	440

Table 5.4: Simplified simulation results for the HDPA and ADE HDPA. The table lists the standard deviation of the number of iterations required for different Environments.

5.3 Chapter Summary

In this chapter, we have discussed our novel contributions to the field of VSSA. Our first contribution is the HDPA, which significantly reduces the number of iterations required before convergence when we have a large action space, and where we impose high accuracy requirements for the converged solution. The HDPA is better than the HCPA for high accuracy requirements because of the delicately designed combination of VSSA, discretization, the Pursuit concept, and structure. Further, in the development of the HDPA, we discovered that the *ordering* of the actions at the leaf level of the hierarchical tree, impacts the performance of the HDPA. Consequently, we proposed the ADE HDPA. The ADE HDPA represents our second contribution to the field of VSSA. These innovations are crucial contributions to the research field, and represent the state-of-the-art solutions. We believe that these results could hopefully remain to be the state of the art in LA, for a few decades!

Chapter 6

Communication-Based Novel Applications

Since the first cellular-phone system was introduced in 1978 [82], and even long before that[1], mobile radio communications have been an important and interesting research domain, worldwide. The development within mobile technology and associated infrastructure over the years is undoubtedly impressive, and research has played a significant part in the evolution of mobile radio communications [82, 83, 84]. The development is still highly prosperous, and researchers are working with advancements to improve the technology even further by, e.g., increasing the speed, increasing the efficiency, and reducing the resource utilization of these systems. The domain of mobile communications represents a compelling field of research due to its dynamic nature and its profound impact in our daily lives.

In recent years, the number of devices utilizing mobile communication networks has been increasing. According to the statistics in the Ericsson's Mobility Report from November 2022, in 2028 there will be 5 billion 5G subscriptions and 9.2 billion mobile subscriptions in total, worldwide [85]. Ericsson also reports that the number of Internet of Things (IoT) connections will grow from 13.2 billion in 2022 to 34.7 billion[2] in 2028. Consequently, the demands for wireless capacity are increasing, and it is mandatory that we have solutions that can accommodate these increasing requirements. The NOMA paradigm is a promising technique to meet these requirements [53], because more users can be multiplexed together in the same RB [57]. The NOMA technology takes advantage of the channel condition differences between the users in the mobile communication systems, which is feasible through the process of Successive Interference Cancellation (SIC) [56]. In simple terms, more users can use the same frequency, but with different power levels based on their individual channel conditions.

[1] In [82], the cellular idea is said to have been introduced in the mid to late 1940s, but to establish exactly when scientists started researching this technology is not the aim of this thesis. Our aim with mentioning these historic happenings, is only to highlight the importance of the field, as a whole.

[2] We emphasize that "standard" mobile phones are not the only devices that can utilize mobile communication networks. Numerous other devices can also utilize such networks. However, in our subsequent discussions, when we discuss "*users*", we refer to mobile phone users.

In NOMA down-link operations, users are grouped into different RBs. For successful SIC, the users to be grouped need to be carefully selected, and their power levels need to be adjusted accordingly [56]. Consequently, the performance in NOMA systems is highly dependent on the user grouping and the power allocation. Therefore, the present aim is to propose a solution for user grouping and power allocation in NOMA systems, and our proposed solution for user grouping was the first reported RL solution to this problem at the time when we submitted the reports of the research for publication. While we did observe that other existing solutions often assumed a particular distribution for the communication channel, the user grouping and power allocation were traditionally carried out and optimized for a static or instantaneous environment. In our research, we proposed to utilize the OMA for the purpose of user grouping with a heuristic approach for the power allocation in a random environment. The benefit with the RL-based OMA solution is that it can group the users over time in an adaptive manner.

This chapter offers an overview of NOMA and our solutions to the user grouping and power allocation problem in a more informal manner than the more detailed explanations presented in this dissertation's accompanying papers. Before explaining our suggested solutions and the results that we have, we will first explain the motivation for tackling the challenging task in the NOMA domain.

6.1 Motivation for this Study

From the introduction above, it is clear that there is an interest in developing better solutions to the domain of mobile communications. Furthermore, frequencies are a finite resource. Thus, because the frequency spectrum is limited, we need more frequency-efficient technology and more effective solutions for frequency sharing. The NOMA technology is more resource effective because multiple mobile users can share the same RB, and therefore, it is a promising technology for the future. As mentioned, the performance in NOMA systems highly depends on which users are grouped together. Consequently, the problem of user grouping in NOMA systems is an important aspect which can be optimized in order to increase the overall efficiency of such systems. However, in our research, we observed that in the existing solutions, the stochastics of the users was not adequately considered. With our proposed OMA approach to the grouping issue in NOMA systems, the users and groupings are monitored over time, where we consider the users' randomness and variations in the channel.

In the field of mobile communications, it is a common practice to assume a certain distribution for the fading of a channel. One example is the Rayleigh fading model. It is comparable to assuming that the stochastic process follows a random and stationary process when the channel coefficient, h, is expected to follow a given time-invariant distribution. User grouping and power allocation have traditionally been carried out based on an instantaneous sample from the assumed distribution. As a result, in most cases, a constant, h, was assumed to have been known and

utilized for the optimization. Thus, in previous solutions, even though channel fading was assumed to be following a certain random distribution, the resulting optimization was based on a rather static nature, not addressing the behavior of the users and their channel characteristics over time. In other words, the channels' stochastic behavior was not considered during the previous power allocation and grouping procedures. Therefore, we wanted to propose new approaches to these issues that consider their behaviors over time, and adjusts the groupings and power levels accordingly.

Channel sounding can be a rather in-expensive operation. However, monitoring the instantaneous channel often enough to accommodate all changes in a real-time manner is practically impossible. Consequently, when a single channel sounding has been conducted, it might be misleading to optimize the system merely based on the most recent channel sounding. Moreover, the general statistics of the channel may change over time for various reasons, including, e.g., movement, moving obstacles, and air movements. Therefore, it can be statistically smarter to have a solution that monitors the behavior over time and accommodates the stochastic changes, e.g., using the OMA. As mentioned above, most earlier solutions to the grouping and power allocation problem optimize the system based on a single channel sounding. Furthermore, due to the complexity of optimization issues, it might not be possible, or preferable, to solve the optimization problem regularly based on, for example, instantaneous channel-sounding results whenever they are available. As a result, we observed that the field needed a more appropriate foundation for the channel coefficients and optimization. In addition, we seek a solution that is adaptive and computationally effective.

6.2 Principles for NOMA and the OMA

The NOMA methodology for mobile radio communication has attracted great attention [53, 54, 57], and the grouping and the power allocation problems in such systems are major concerns because these realizations highly impact the overall performance and throughput in the mobile communication system. The concept of NOMA is that the users can be multiplexed together into the same RB. In Figure 6.1, we visualize the difference between conventional Orthogonal Multiple Access and NOMA [86]. In the figure, we see that more users can share the same RB in the NOMA case, while they are distinctly separated onto different orthogonal resources in the conventional case [86]. The sharing implies that users in different traffic classes can transmit concurrently on the same RB to enhance latency and fairness [87]. More specifically, in conventional Orthogonal Multiple Access, the users are divided into different orthogonal frequencies, while they can be super-positioned into the same frequency resource with different power allocations in the NOMA case.

Let us briefly address some of the existing research related to grouping and power allocation issues in NOMA. In [88], the authors introduced a fairness power allocation algorithm to address the power issue in NOMA systems. The authors in [89],

investigated the sum-rate and outage probability in down-link NOMA systems. The up-link scenario has also been investigated, with a power back-off method proposed in [90]. These aforementioned research initiatives mainly focused on intra-cell interference, where the interference is generated by the users within the same group. As opposed to this, the inter-cell interference occurs between groups within the NOMA system. Inter-cell interference was the topic of [91], where the authors considered and evaluated both down-link and up-link scenarios in a dense multiple cell NOMA network. In the NOMA paradigm, mmWave networks have also been investigated, as in [92] and [93], where they considered beam-forming options, without taking into account the users' locations. Further, the "beamforming" strategy and power allocation were jointly optimized for maximizing the throughput in [94]. In [57], the authors used maximum weight matching to build disjoint groups of users, allocated one RB to each group, and then explained a method for allocating power inside the groups. Exhaustive search based on proportional fairness for solving the grouping problem was presented in [58]. In both [54] and [95], the authors considered groups of cardinality two, where the former invoked the so-called Hungarian algorithm, and the latter considered proportional fairness for power allocation. In terms of ML in the NOMA domain, the study was still in its infancy when we addressed the grouping using the OMA-based approach. However, one example was the use of K-Means for addressing the grouping in NOMA systems [56], where the authors proposed utilizing K-Means to obtain clusters based on the users' geolocation, demonstrated by a *school hall* example scenario.

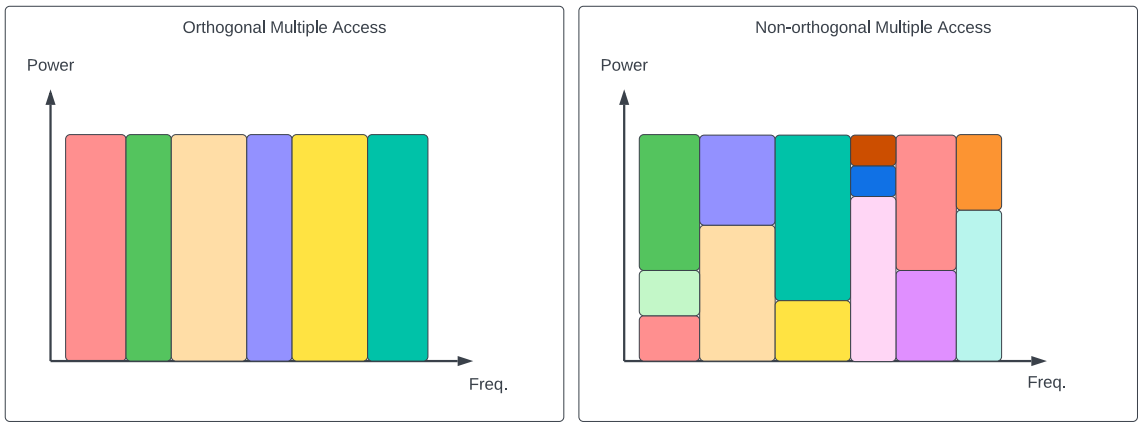


Figure 6.1: The figure visualizes the difference between the orthogonal multiple access and NOMA methods. In the example on the left side of the figure, the users have different orthogonal resources. On the contrary, they are super-positioned and allocated different power levels in the example on the right side of the figure.

In our proposed grouping and power allocation approaches, we considered a simplified single-carrier down-link system. We aimed to group the K users in the mobile system into N distinct and non-overlapping groups. In Figure 6.2, we visualize the concept. On the left side of the figure, we have a NOMA system with one BS, and six users to be grouped into three groups. On the right side of the figure, we see that a grouping of the users has been obtained. In the research cited above, the

authors did not explore the stochastic nature of wireless communication channels, and the channel characteristics were assumed to have been known *a priori*. When the researchers assume a channel characteristic (h), optimization results can only be valid when the coefficients are reasonably close to the actual values. Consequently, with our proposed grouping solution, we aimed to obtain a grouping, as visualized on the right side of the figure, while considering the mobile users by being able to handle their channel characteristics, i.e., h, changing over time. Therefore, the mobile users in our simulations were configured to move within a defined area, i.e., like the behavior of users in an university or an office building.

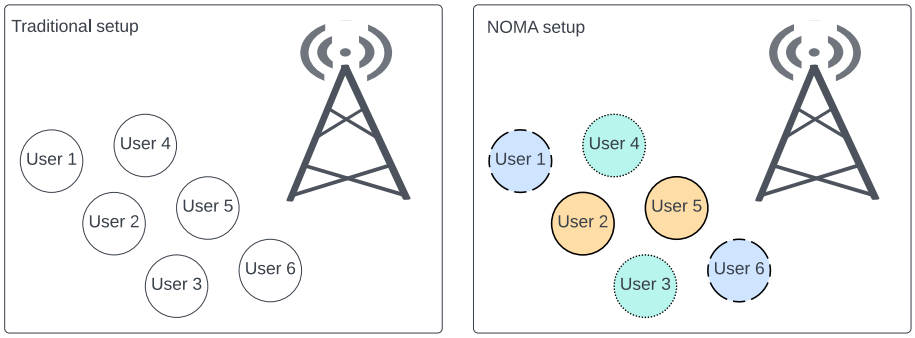


Figure 6.2: A visualization of the NOMA concept, where the users are grouped in the box on the right side.

In Figure 6.3, we visualize the concept of NOMA [87]. The figure displays a down-link NOMA communication scenario when two users are grouped together in the system. Let us assume that in the figure, User 2 has better channel quality when compared with User 1. Through the NOMA concept, the user with *good* channel quality (User 2) can eliminate interference from the user with poor channel quality by using SIC. In contrast, the user with poor channel quality (User 1) decodes its signal without using SIC[3]. Hence, in the visualization, the process for User 2 has two steps, and the process for User 1 has a single step. In this way, User 1 and User 2 can utilize the same RB instead of requiring two different ones, given that SIC can be performed adequately. When we have more than two users in the same group, the same concept applies, where a user can remove the signal components from users with lower channel coefficients in its group via SIC and then retrieve its message in a step-wise manner. Consequently, the problem becomes that of knowing which users must be grouped, and how to adjust their power for successful and fair NOMA operations.

In our new OMA-based solution to the user grouping in NOMA systems, we proposed to utilize the EOMA variant. However, any variant within the OMA paradigm can also be used for this scenario. The utilization of the GCD and PSR implementation can also be investigated as part of future work. The concept of using OMA for this purpose, is to group and monitor the established groupings over

[3]For a user to perform SIC, there is a requirement to the channel characteristics of the specific user and to the users inside its group, which needs to be fulfilled. The attached NOMA paper details the SIC requirement and is not given here.

time, handling their stochastic behavior and adjusting the groups accordingly. The OMA concept is based on queries presented to the automaton over time. Based on learning through a reinforcement Reward and Penalty principle, the objects to be grouped, i.e., the users in the NOMA scenario, traverse the states of the automaton, and the current action that they reside in determines their group. Consequently, the NOMA grouping problem needs to be configured in such a way that the OMA algorithms are able to solve it, which will be addressed in more detail below[4].

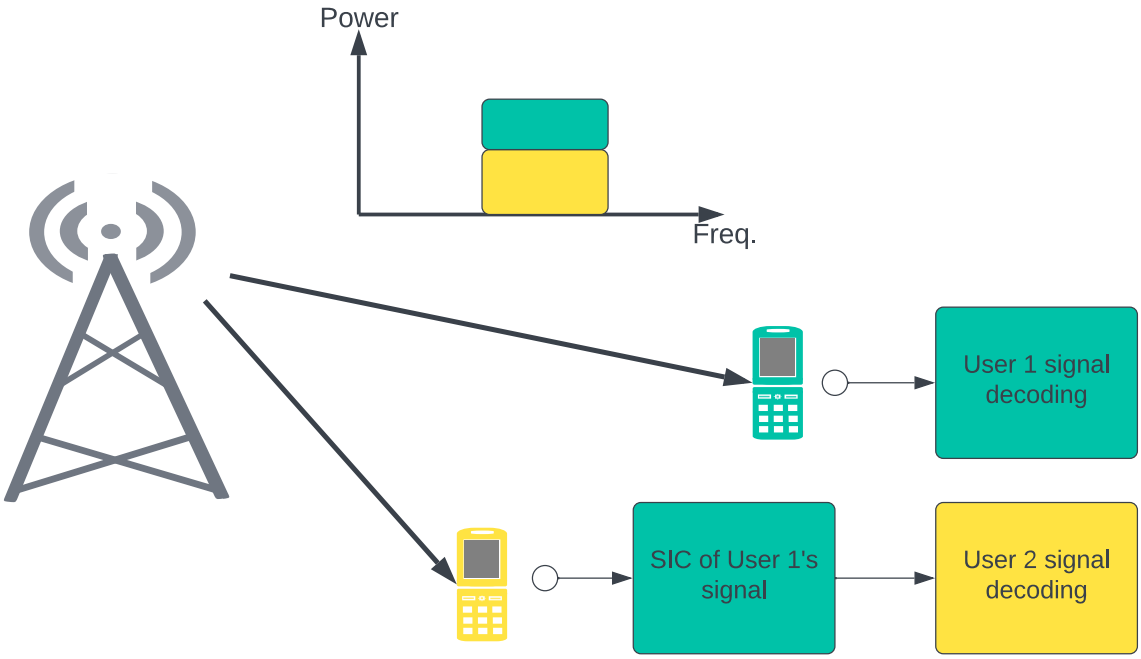


Figure 6.3: A single-carrier down-link system with two users and one BS.

The second contribution to the field of NOMA in our research, is related to the power allocation issue in such systems. More specifically, we proposed two heuristic-based approaches for power allocation once the groups have been established within the NOMA system. In simple terms, the first approach was a greedy algorithm, and the second one was an algorithm that considered the users more fairly based on their channel characteristics. In the greedy algorithm, we allocate the minimum required power to fulfill the data rate needed for the weakest users in the system, and give the remaining power to the user with the best channel coefficient, ensuring that the joint data rates (E) is maximized[5]. However, the greedy approach can result in highly imbalanced data rates among the users, which can be unfair. Therefore, the second approach allocates the power more equitably, according to a relation between the users' channel coefficients, such that the power level is adjusted more evenly between the users in the system. The power allocation approaches constitute solutions to allocate the power within a group after the establishment of the group,

[4]We have detailed the algorithms in the OMA domain in earlier parts of this dissertation. Therefore, the algorithms will not be repeated in this chapter.

[5]In the paper about NOMA that is presented later in this dissertation, we used R to refer to the data rate. However, in this thesis we have used R to refer to the number of actions. Therefore, in this chapter, we utilize E to report the data rate.

and the power allocation process needs to be conducted for all the groups within the NOMA system. The reader should note that we only considered the intra-cell interference for our approaches and that further work should also consider the inter-cell interference. We refer to the two approaches as the *greedy algorithm* and the *channel coefficient-based algorithm*, respectively.

6.3 Algorithm Summary

As a part of the research contributions of this dissertation, we proposed a novel OMA-based solution to the grouping problem in NOMA systems. In addition, we introduced two approaches to solve the power allocation problem associated with the users that are grouped in such systems. In this section, we explain the principles of our OMA approach in NOMA systems, and our related power allocation methodologies. We explain the concepts in a brief manner[6]. The proposed solution is two-pronged. The first prong concerns the grouping of users, and the second prong concerns allocating power within the obtained groups. We will present these two concepts subsequently.

6.3.1 OMA for User Grouping

We proposed to use an OMA-based approach to solve the grouping problem in NOMA systems. The reader should note that we can use any of the algorithms in the OMA family for the grouping purpose. However, we described using EOMA as an example in our NOMA papers, and we will also use the EOMA as the example algorithm here. The EOMA can learn by itself, without prior knowledge of the system parameters, channels, or clusters in this particular case, by interactions with the Environment over time. In the NOMA case, our Environment is the communication system, and the objects to be grouped are the users in the NOMA system.

When we initialize the EOMA algorithm, the users are represented as abstract objects and distributed equally among the boundary states in the EOMA[7]. The reader should note that the state depth can be adjusted according to our preferences. A large state space per action (cluster) makes the algorithm more accurate. However, the number of iterations before convergence increases when the state depth increases. For K users to be grouped into N groups in the NOMA system, we need $\frac{K}{N} = L_c$ clusters in the EOMA, where N and L_c are integers. Based on the corresponding state of the object, we can infer the cluster of the corresponding user, and through interactions with the Environment, these objects are moved around the state space of the automaton upon receiving rewards or penalties. The goal of the EOMA is to cluster users with similar ranking categories over time while handling the stochastics.

[6]We refer the reader to the NOMA paper presented later in this dissertation for the finer details about the proposed solutions.

[7]As already explained in this dissertation, the existing EOMA could only handle groups of equal sizes, which we considered in our proposed solution to the grouping in NOMA systems. However, thereafter, we proposed the GCD and PSR variants, which can also be utilized for the grouping purpose.

After the EOMA algorithm is initialized, the BS, or another processing unit, conducts a channel sounding, and based on the results, it ranks all the users from 1 to K. After that, we generate queries, and if the two users that are randomly uniformly combined to a query are within the same *ranking category*, they are rewarded if they are currently clustered in the EOMA, and they are penalized if they are not. The ranking categories are defined such that we have L_c ranking categories, and the worst channel qualities are given for the ranks in the ranking list from 1 to N, and the next ranking category is for the ranks in the ranking list from $1+N$ to $2N$, and so on. The user with the smallest h value has the system's worst channel conditions and the first position in the ranking list. The channel sounding, ranking, and query generation is continued until the EOMA has converged, or we can use the current clustering in an online manner. The ranking can be based on instantaneous values or averages over time, i.e., Δt. Our work mainly focused on the averaged values of h for the ranking procedure. However, the basis for the grouping can be adjusted to be based on other parameters that characterize the uses as well.

The EOMA adaptively determines the users that are similar and clusters them. After the users have been clustered, we need to invoke the last part of the proposed process, where we obtain the final groups that constitute the basis for the power allocation. In this latter phase, the users are grouped based on their ranking within the respective groups. The users with the same place in the ranking list from each cluster are put in the same group. More specifically, we have L_c clusters, with N users in each cluster. These users are ranked from 1 to N inside each cluster, and we put the users with the same ranking into the final groups. The overall grouping process is visualized in Figure 6.4. Hence, queries are sent to the EOMA based on the users' channel characteristics and ranking, and the EOMA's obtained clusters are utilized by the latter grouping phase, where the final groups are determined. The grouping process could be carried out in the BS or another processing unit.

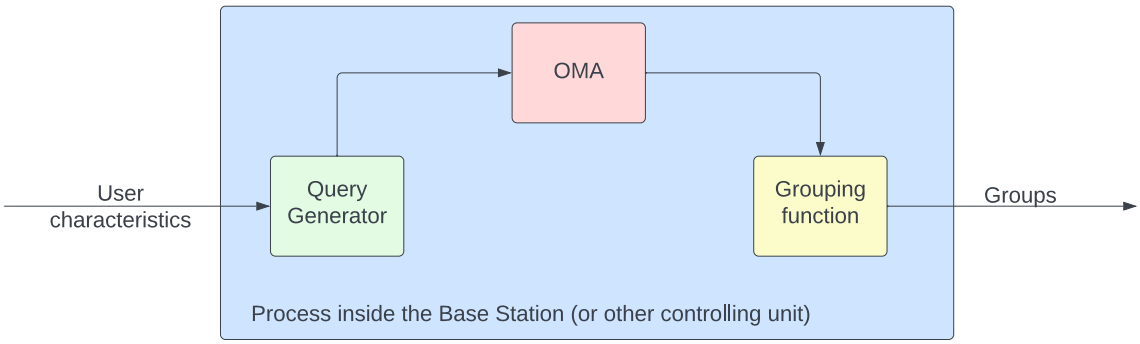


Figure 6.4: A visualization of the OMA-based NOMA operation.

6.3.2 Heuristics for Power Allocation

When the users are grouped, the power must be properly allocated to the different users. The reader should note that in mobile radio communication systems, authorities usually define thresholds to the maximum power levels allowed in the system. Let us assume that the maximum power level for a group is given by E_i for

group i in the NOMA system. In our first power allocation approach, we wanted to maximize E while fulfilling a QoS requirement for all of the users. Thus, all users should be allocated more than zero power, and the sum of the given power to all users should be equal to, or below, a maximum power threshold. Consequently, we want to determine the power of all the different users within each group to maximize the total data rate of all groups in the NOMA system.

In the greedy algorithm, we can solve the problem outlined in the paragraph above by providing the excess power to the user with the highest channel-coefficient, and we distribute just enough power to the users with a smaller h to meet the minimum needed data rate. We allocate the majority of power to the users with higher channel coefficients, resulting in the highest obtainable sum rate for the system. With the greedy algorithm, the users with good channel quality are benefited more than the users with poorer channel quality while fulfilling the QoS requirement.

As mentioned earlier, the greedy algorithm has the drawback that the data rates between the users can become highly unbalanced. More specifically, a strong user can obtain most of the power, resulting in a high data rate, while a weaker user only experiences the QoS-required data rate. To mitigate this issue, we proposed the channel coefficient-based algorithm. The channel coefficient-based algorithm is based on a relation between the users' $|h|^2$ values. Consequently, when users are part of an obtained group, their power levels are determined proportionally to their channel qualities in relation to one another. We considered the Signal Interference Noise Ratio (SINR), intra-group interference, and the data rate in this approach for optimizing the sum of the data rates in the NOMA system.

The concept of the channel coefficient-based algorithm is that we first allocate power to the weakest user[8]. By setting the weakest user's data rate to the required QoS data rate, we can calculate the power needed for that user. After that, we need to find the power of the users between the weakest and strongest users. We find these power levels by using the SINR of the previous user multiplied by the relation between the h value for the previous user and the user we are calculating the power level for. Once we have the power levels for the users between the weakest and the strongest user, we can allocate the remaining power to the strongest user. In this way, the power is allocated in a more balanced manner between the users based on their channel quality realized through SINR and h.

6.4 Overview of the NOMA-based Results

This section summarizes some of the results obtained for our proposed approaches to user grouping and power allocation in NOMA systems. In our experiments, we utilized MatLab to simulate the behavior of h for the different users. We utilized a Rayleigh distribution as the basis for the values of h, with a carrier frequency

[8]Whether a user is a *strong* or *weak* user is determined by its h value. Thus, the higher the value of h, the better the channel quality. The characteristic of being weak or strong is seen in relation to the others' h values.

of 5.7 GHz. As mentioned, we implemented mobility into our simulations corresponding to the movement inside an office building. Specifically, we used a mobility factor of 2 $\frac{km}{h}$ for the users' receivers. Moreover, we sampled the values of h for the different users according to $\frac{1}{2f_d}$, where f_d is the Doppler frequency, and $f_d = f_c(\frac{v_U}{v_L})$, where f_c is the carrier frequency, v_U is users' mobility factor, and v_L is the speed of light. With the aforementioned configurations, we could simulate Environments with different numbers of users and channels.

To demonstrate the performance of the EOMA for grouping the users, we based our accuracy on whether the LA obtained the clusters that corresponded to the minimal difference between the users in the cluster. We evaluated the intra-group similarity based on the users' configured mean values of h in our simulations. The EOMA algorithm was provided with pairwise inputs of users and it discovered the correct grouping in 100% of the experiments for experiments with $-30dB$ difference between the average values of $h(t)$ for groups of size $4, 6, 8, 10, 12, 14, 16, 18$ and 20, with two users in each group[9]. In the EOMA, we used eight states, and for the simulations, we averaged the performance over 100 experiments per group size configuration. For the user grouping with our EOMA-based solution, we also demonstrated that the algorithm could follow the change in the channel characteristics adaptively. To cite one example, where two users distinctly changed channel characteristics after around 40 samples, we visualized that the EOMA detected the change, and made the users change group approximately 20 samples after the change was detected.

To demonstrate the performance of the power allocation methods, we compared them to the sum data rate obtained by an exhaustive grid search. We implemented the exhaustive grid search with a step size of 0.001 ($\gamma = 0.001$), and we carried out the exhaustive search for the same groups and values obtained of h for the greedy and the channel coefficient-based solutions. We tested the methods for instantaneous values of h and time averages of five samples. We compared the exhaustive search, greedy, and channel coefficient-based algorithms. Our results demonstrated that the greedy and exhaustive search algorithms obtained the same overall data rate and that the channel coefficient-based solution obtained a lower sum rate. However, the channel coefficient-based solution ensured increased fairness in the system, with a more balanced power distribution among the users. Hence, there is a trade-off between the system's sum rate and the users' fairness. In comparison, the exhaustive search required testing $\left(\frac{E_i}{\gamma}\right)^{L_c}$ combinations, where L_c is the number of users inside a group, the greedy and channel coefficient-based solution required $2L_c$ computations. Hence, our proposed solutions required less computation.

Because the channel characteristics and stochastics of the users can be adaptively tracked over time, it is clear that the OMA approach for the grouping and the two heuristics for the power allocation, represent practical solutions to the grouping and power allocation problems in NOMA systems. This is especially true from the perspective of the results that have been summarized above, which make it clear that the proposed approaches represent effective state-of-the-art solutions.

[9]The details of these experiments can be seen in more detail in the NOMA paper attached to this dissertation.

6.5 Chapter Summary

In this chapter, we have highlighted the importance of improving the efficiency and applicability of solutions in the mobile radio communication domain. We have briefly explained the concept of NOMA. Furthermore, we explained our proposed approach for solving the grouping and power allocation problems in NOMA systems. More specifically, we proposed the EOMA variant in the OMA paradigm to group and monitor the users over time in NOMA systems. Our simulation results, which we briefly highlighted above, demonstrated a 100% accuracy for finding the clusters with similar $h(t)$ over time, also in changing environments. Thus, our approach to the grouping problem considers the stochastic nature of the users' channel coefficients as the system evolves. For power allocation in NOMA systems, we proposed two solutions for maximizing the sum rate with a user QoS constraint. The reader should note that the grouping and power allocation can be used independently (or combined) in a NOMA system, making our proposed approach adaptive and configurable to different scenarios. In conclusion, this chapter encapsulates our contributions to the practical application of NOMA systems.

Chapter 7

Conclusion

In the Ph. D. work summarized in this dissertation, we have explored the intriguing field of LA, and its applications in various domains. One of the major benefits of LA is that these machines are powerful tools in stochastic and dynamic Environments. LA, which possess well-established mathematically-proven properties, can solve complex problems through learning. Throughout this study, we have delved into the finer details of some of the algorithms within this domain. We have made significant contributions to the field, where we enhanced both the applicability and performance of different state-of-the-art algorithms within the field of LA, and explored practical implementations of LA algorithms for solving problems in the domain of mobile radio communication.

The first essential contribution of our research concerned the development of novel approaches to the family of OMA algorithms. The LA-based OMA algorithms can be used to solve partitioning problems. However, the existing OMA algorithms could only solve problems with equally sized groups. Consequently, we proposed two new variants, namely the GCD and PSR OMA variants. By exploiting the adaptive and exploratory behavior of LA, the GCD and PSR variants make it possible for OMA algorithms to solve NEPPs, i.e., grouping problems where the group sizes can be both equal and *un-equal*. Our experimental results demonstrated that these algorithms can solve NEPPs effectively and accurately, where the PSR variant is superior to the GCD variant in terms of average number of iterations required before convergence for high noise levels. In addition, we summarized the entire field of OMA, its potential benefit to society, and outlined the further work for future researchers in the field.

All of these results have yielded, referenced publications, that are included in the thesis.

Our next key research contribution concerned hierarchical LA, where we proposed the HDPA and the ADE approach, with mathematical proofs formalizing their behavior. Our experimental results clearly demonstrated the superior performance of the HDPA for high accuracy requirements when compared with the previous state-of-the-art algorithm, namely the HCPA. The ADE approach that has also been proposed and implemented, concerns the ordering of the actions at the leaf level in hierarchical LA, and can improve the efficiency of these algorithms by reduc-

ing the number of iterations required before convergence. Our experimental results demonstrated that, e.g., the ADE HDPA is superior when compared to the plain HDPA variant. Consequently, these innovations to the VSSA type of LA represent, to the best of our knowledge, the state of the art.

Furthermore, we have proposed the use of LA for solving the intricate grouping problem in NOMA systems, where an OMA-based approach was suggested for partitioning and monitoring the stochastic users in NOMA systems. The OMA solution represents an adaptive solution that can accommodate the changes in user's behaviors over time. This phenomenon has been demonstrated through simulations. In addition, we proposed two heuristics for allocating the power in NOMA systems once a grouping has been attained. The solution to the user grouping and the solutions to the power allocation problem are standalone, and they can be used independently or combined in NOMA systems, making them versatile contributions to the field.

If we look into the future, numerous challenges can be further investigated in the field of LA. For example, in the case of the OMA, the mathematical analysis of its convergence is open, and the issue of considering queries consisting of more than two objects, is unsolved, and these are important aspects for future research. Furthermore, increasing the speed and investigating hierarchical structures in the OMA paradigm are other examples of potential further work. In the case of VSSA, with hierarchical structures, an interesting possibility to investigate is the hierarchical LA's potential in classification problems, where multiple LA would cooperate in a forest-like structure with majority voting principles, which would be an area that is tightly intertwined with *game theory*. Furthermore, there are numerous applications where LA can represent versatile and adaptive solutions to stochastic problems. Consequently, studying applications for LA is also a thread that can lead to many improvements and possible new insights into several domains.

In conclusion, this Ph. D. study has demonstrated the potential of LA as a tool and paradigm in ML, and as an adaptive computational tool. The research and work conducted within this study has contributed to the theoretical understanding and algorithm development of LA, and demonstrated its use in a practical application. We believe that our findings and the dissemination of these results will inspire further research and innovations, and hopefully, push the field even further. With continued interest, exploration and developments, LA can undoubtedly play a vital part in solving complex problems, and in the evolution and future of intelligent systems.

Bibliography

[1] Sondre Glimsdal and Ole-Christoffer 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*, pages 594–599. IEEE, December 2014.

[2] Anis Yazidi, Xuan Zhang, Lei 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*, 31(2):512–526, February 2020.

[3] Tsetlin. *Automation Theory and Modeling of Biological Systems*. Academic Press, February 1974.

[4] K. S. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction*. Dover Publications, Mineola, New York, illustrated edition, December 2012.

[5] B. John Oommen, Xuan Zhang, and Lei Jiao. A Comprehensive Survey of Estimator Learning Automata and Their Recent Convergence Results. In Petros Nicopolitidis, Sudip Misra, Laurence T. Yang, Bernard Zeigler, and Zhaolng Ning, editors, *Advances in Computing, Informatics, Networking and Cybersecurity: A Book Honoring Professor Mohammad S. Obaidat's Significant Scientific Contributions*, Lecture Notes in Networks and Systems, pages 33–52. Springer International Publishing, March 2022.

[6] M. A. L. Thathachar and K. R. Ramakrishnan. A Hierarchical System of Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(3):236–241, March 1981.

[7] S. Lakshmivarahan. *Learning Algorithms Theory and Applications: Theory and Applications*. Springer, softcover reprint of the original 1st ed. 1981 edition, November 1981.

[8] S. Lakshmivarahan and M. A. L. Thathachar. Absolutely Expedient Algorithms for Stochastic Automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(3):281–286, May 1973.

[9] M. A. L. Thathachar and P. S. Sastry. Estimator Algorithms for Learning Automata. In *Proc. Platinum Jubilee Conf. Syst. Signal Process., Bangalore, India*, pages 29–32, December 1986.

[10] B. John Oommen and J.K. Lanctot. Discretized Pursuit Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(4):931–938, July 1990.

[11] 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)*, 31(3):277–287, June 2001.

[12] M. Agache and B. John Oommen. Generalized Pursuit Learning Schemes: New Families of Continuous and Discretized Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 32(6):738–749, December 2002.

[13] Xuan Zhang, Ole-Christoffer Granmo, and B. John Oommen. The Bayesian Pursuit Algorithm: A New Family of Estimator Learning Automata. In Kishan G. Mehrotra, Chilukuri K. Mohan, Jae C. Oh, Pramod K. Varshney, and Moonis Ali, editors, *Modern Approaches in Applied Intelligence*, Lecture Notes in Computer Science, pages 522–531. Springer, June 2011.

[14] Xuan Zhang, Ole-Christoffer Granmo, and B. John Oommen. Discretized Bayesian Pursuit – A New Scheme for Reinforcement Learning. In He Jiang, Wei Ding, Moonis Ali, and Xindong Wu, editors, *Advanced Research in Applied Artificial Intelligence*, Lecture Notes in Computer Science, pages 784–793. Springer, June 2012.

[15] Xuan Zhang, Ole-Christoffer Granmo, and B. John Oommen. On Incorporating the Paradigms of Discretization and Bayesian Estimation to Create a New Family of Pursuit Learning Automata. *Applied Intelligence*, 39(4):782–792, December 2013.

[16] A. Shirvani and B. John Oommen. On Enhancing the Object Migration Automaton Using the Pursuit Paradigm. *Journal of Computational Science*, 24:329–342, January 2018.

[17] A. Shirvani and B. John Oommen. On Enhancing the Deadlock-Preventing Object Migration Automaton Using the Pursuit Paradigm. *Pattern Analysis and Applications*, 23(2):509–526, May 2020.

[18] 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)*, pages 295–302, October 2017.

[19] G.I. Papadimitriou. Hierarchical Discretized Pursuit Nonlinear Learning Automata with Rapid Convergence and High Accuracy. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):654–659, August 1994.

[20] M. A. L. Thathachar and P. S. Sastry. A Hierarchical System of Learning Automata That Can Learn die Globally Optimal Path. *Information Sciences*, 42(2):143–166, July 1987.

[21] M. A. L. Thathachar and P. S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Springer, New York, September 2011.

[22] A. V. Vasilakos, N. H. Loukas, and A. F. Atlasis. The Use of Learning Algorithms in ATM Networks Call Admission Control Problem: A Methodology. In *Proceedings of 20th Conference on Local Computer Networks*, pages 407–412, October 1995.

[23] A. F. Atlasis and A. V. Vasilakos. The Use of Reinforcement Learning Algorithms in Traffic Control of High Speed Networks. In Hans-Jürgen Zimmermann, Georgios Tselentis, Maarten van Someren, and Georgios Dounias, editors, *Advances in Computational Intelligence and Learning: Methods and Applications*, International Series in Intelligent Technologies, pages 353–369. Springer Netherlands, 2002.

[24] A. V. Vasilakos, M.P. Saltouros, A.F. Atlasis, and W. Pedrycz. Optimizing QoS Routing in Hierarchical ATM Networks Using Computational Intelligence Techniques. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 33(3):297–312, August 2003.

[25] M. R. Meybodi and H. Beigy. New Learning Automata Based Algorithms for Adaptation of Backpropagation Algorithm Parameters. *International Journal of Neural Systems*, 12(1):45–67, February 2002.

[26] C. Unsal, P. Kachroo, and J. S. Bay. Simulation Study of Multiple Intelligent Vehicle Control using Stochastic Learning Automata. *Transactions of the Society for Computer Simulation International: Special Issue On Simulation Methodology in Transportation Systems*, 14(4):193–210, December 1997.

[27] C. Unsal, P. Kachroo, and J. S. Bay. Multiple Stochastic Learning Automata for Vehicle Path Control in an Automated Highway System. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 29(1):120–128, January 1999.

[28] M. Barzohar and D.B. Cooper. Automatic Finding of Main Roads in Aerial Images by Using Geometric-stochastic Models and Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):707–721, July 1996.

[29] Christian Bettstetter, Hannes Hartenstein, and Xavier Pérez-Costa. Stochastic Properties of the Random Waypoint Mobility Model. *Wireless Networks*, 10(5):555–567, September 2004.

[30] B. John Oommen and J. R. Zgierski. Breaking Substitution Cyphers Using Stochastic Automata. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(2):185–192, February 1993.

[31] B. John Oommen and Chris Fothergill. Fast Learning Automaton-Based Image Examination and Retrieval. *The Computer Journal*, 36(6):542–553, January 1993.

[32] B. John Oommen Anis Yazidi, Ole-Christoffer Granmo. Service Selection in Stochastic Environments: A Learning-Automaton Based Solution. *Applied Intelligence*, 36(3):617–637, April 2012.

[33] A. Jobava. Intelligent Traffic-aware Consolidation of Virtual Machines in a Data Center. Master's thesis, University of Oslo, Norway, May 2015.

[34] F. M. Ung. Towards Efficient and Cost-Effective Live Migrations of Virtual Machines. Master's thesis, University of Oslo, Norway, May 2015.

[35] Meybodi M. R. Mamaghani A. S., Mahi M. 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*, pages 8–12. IEEE, June 2010.

[36] E. Fayyoumi and B. John Oommen. A Fixed Structure Learning Automaton Micro-aggregation Technique for Secure Statistical Databases. In Josep Domingo-Ferrer and Luisa Franconi, editors, *Privacy in Statistical Databases*, Lecture Notes in Computer Science, pages 114–128. Springer, 2006.

[37] 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)*, 39(5):1192–1205, October 2009.

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

[39] E. Bisong. On Designing Adaptive Data Structures with Adaptive Data "Sub"-Structures. Master's thesis, Carleton University, Canada, October 2018.

[40] Ole-Christoffer Granmo. The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic. arXiv, April 2018. arXiv:1804.01508 [cs].

[41] Geir Thore Berge, Ole-Christoffer Granmo, Tor Oddbjørn Tveit, Morten Goodwin, Lei Jiao, and Bernt Viggo Matheussen. Using the Tsetlin Machine to Learn Human-Interpretable Rules for High-Accuracy Text Categorization with Medical Applications. *IEEE Access*, 7:115134–115146, August 2019.

[42] Bimal Bhattarai, Lei Jiao, and Ole-Christoffer Granmo. Measuring the Novelty of Natural Language Text Using the Conjunctive Clauses of a Tsetlin Machine Text Classifier. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pages 410–417. INSTICC, SciTePress, 2021.

[43] Rupsa Saha, Ole-Christoffer Granmo, and Morten Goodwin. Mining Interpretable Rules for Sentiment and Semantic Relation Analysis using Tsetlin Machines. In Max Bramer and Richard Ellis, editors, *Artificial Intelligence XXXVII*, Lecture Notes in Computer Science, pages 67–78, Cham, December 2020.

[44] Rohan K. Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. Human-Level Interpretable Learning for Aspect-Based Sentiment Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(16):14203–14212, May 2021.

[45] Rohan K. Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. Interpretability in Word Sense Disambiguation using Tsetlin Machine. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*, pages 402–409. SCITEPRESS - Science and Technology Publications, 2021.

[46] Lei Jiao, Xuan Zhang, Ole-Christoffer Granmo, and K. Darshana Abeyrathna. On the Convergence of Tsetlin Machines for the XOR Operator. arXiv, January 2021. arXiv:2101.02547 [cs].

[47] Xuan Zhang, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. On the Convergence of Tsetlin Machines for the IDENTITY- and NOT Operators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6345–6359, October 2022.

[48] Xuan Zhang, Ole-Christoffer Granmo, B. John Oommen, and Lei Jiao. A Formal Proof of the ϵ-Optimality of Absorbing Continuous Pursuit Algorithms Using the Theory of Regular Functions. *Applied Intelligence*, 41(3):974–985, October 2014.

[49] Xuan Zhang, B. John Oommen, Ole-Christoffer Granmo, and Lei Jiao. A Formal Proof of the ϵ-Optimality of Discretized Pursuit Algorithms. *Applied Intelligence*, 44(2):282–294, March 2016.

[50] Xuan Zhang, Lei Jiao, B. John Oommen, and Ole-Christoffer Granmo. A Conclusive Analysis of the Finite-Time Behavior of the Discretized Pursuit Learning Automaton. *IEEE Transactions on Neural Networks and Learning Systems*, 31(1):284–294, January 2020.

[51] Brian T. Mitchell and Dionysios I. Kountanis. A Reorganization Scheme for a Hierarchical System of Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(2):328–334, March 1984.

[52] D.T.H. Ng, B. John Oommen, and E.R. Hansen. Adaptive Learning Mechanisms for Ordering Actions Using Random Races. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(5):1450–1465, September 1993.

[53] Yuanwei Liu, Zhijin Qin, Maged Elkashlan, Zhiguo Ding, Arumugam Nallanathan, and Lajos Hanzo. Nonorthogonal Multiple Access for 5G and Beyond. *Proceedings of the IEEE*, 105(12):2347–2381, December 2017.

[54] Mohammad Ali Sedaghat and Ralf R. Müller. On User Pairing in Uplink NOMA. *IEEE Transactions on Wireless Communications*, 17(5):3474–3486, May 2018.

[55] Yue Yin, Yang Peng, Miao Liu, Jie Yang, and Guan Gui. Dynamic User Grouping-Based NOMA Over Rayleigh Fading Channels. *IEEE Access*, 7:110964–110971, August 2019.

[56] Jingjing Cui, Zhiguo Ding, Pingzhi Fan, and Naofal Al-Dhahir. Unsupervised Machine Learning-Based User Clustering in Millimeter-Wave-NOMA Systems. *IEEE Transactions on Wireless Communications*, 17(11):7425–7440, November 2018.

[57] Mylene Pischella and Didier Le Ruyet. NOMA-Relevant Clustering and Resource Allocation for Proportional Fair Uplink Communications. *IEEE Wireless Communications Letters*, 8(3):873, June 2019.

[58] Xiaohang Chen, Anass Benjebbour, Anxin Li, and Atsushi Harada. Multi-User Proportional Fair Scheduling for Uplink Non-Orthogonal Multiple Access (NOMA). In *2014 IEEE 79th Vehicular Technology Conference (VTC Spring)*, pages 1–5, May 2014.

[59] F. Liu, P. Mähönen, and M. Petrova. Proportional Fairness-Based Power Allocation and User Set Selection for Downlink NOMA Systems. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2016.

[60] Li-Hui Tasi. The Modified Differencing Method for the Set Partitioning Problem with Cardinality Constraints. *Discrete Applied Mathematics*, 63(2):175–180, November 1995.

[61] Mehmet Hacibeyoglu, Vahit Tongur, and Kemal 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)*, pages 1–5, October 2014.

[62] N. Karmarker and R. M. Karp. The Differencing Method of Set Partitioning. Technical Report, University of California, Berkeley, USA, January 1983.

[63] Richard E. Korf. From Approximate to Optimal Solutions: A Case Study of Number Partitioning. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, IJCAI'95, pages 266–272, San Francisco, CA, USA, August 1995.

[64] Richard E. Korf. A Complete Anytime Algorithm for Number Partitioning. *Artificial Intelligence*, 106(2):181–203, December 1998.

[65] P. C. Pop and O. Matei. A Genetic Algorithm Approach for the Multidimensional Two-Way Number Partitioning Problem. In Giuseppe Nicosia and Panos

Pardalos, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 81–86, Berlin, Heidelberg, November 2013.

[66] Jozef Kratica, Jelena Kojić, and Aleksandar Savić. Two Metaheuristic Approaches for Solving Multidimensional Two-Way Number Partitioning Problem. *Computers & Operations Research*, 46:59–68, June 2014.

[67] B. John Oommen and D. C. Y. Ma. Deterministic Learning Automata Solutions to the Equipartitioning Problem. *IEEE Transactions on Computers*, 37(1):2–13, January 1988.

[68] Sondre Glimsdal and Ole-Christoffer Granmo. A Bayesian Network Based Solution Scheme for the Constrained Stochastic On-line Equi-Partitioning Problem. *Applied Intelligence*, 48(10):3735–3747, October 2018.

[69] M. Hammer and Bahram Niamir. A Heuristic Approach to Attribute Partitioning. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD '79, pages 93–101, New York, USA, May 1979.

[70] C. T. Yu, C. Suen, K Lam, and M. K. Siu. Adaptive Record Clustering. *ACM Transactions on Database Systems (TODS)*, 10(2):180–204, June 1985.

[71] D. Ciu and Y. Ma. Object Partitioning by Using Learning Automata. Master's thesis, Carleton University, Canada, April 1986.

[72] B. John Oommen and D.C.Y. Ma. Stochastic Automata Solutions to the Object Partitioning Problem. Technical Report TR-103, Carleton University, Canada, November 1986.

[73] W. Gale, S. Das, and C. T. Yu. Improvements to an Algorithm for Equipartitioning. *IEEE Transactions on Computers*, 39(5):706–710, May 1990.

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

[75] Rebekka Olsson Omslandseter. Learning Automata-Based Object Partitioning with Pre-Specified Cardinalities. Master's thesis, University of Agder, Norway, May 2020.

[76] B. John Oommen. Absorbing and Ergodic Discretized Two-Action Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(2):282–293, March 1986.

[77] B. John Oommen and J.P.R. Christensen. Epsilon-Optimal Discretized Linear Reward-Penalty Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(3):451–458, May 1988.

[78] J.K. Lanctot and B. John Oommen. Discretized Estimator Learning Automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1473–1483, November 1992.

[79] Xuan Zhang, B. John Oommen, and Ole-Christoffer Granmo. The Design of Absorbing Bayesian Pursuit Algorithms and the Formal Analyses of Their ϵ-Optimality. *Pattern Analysis and Applications*, 20(3):797–808, August 2017.

[80] Loïc Lannelongue, Jason Grealey, and Michael Inouye. Green Algorithms: Quantifying the Carbon Footprint of Computation. *Advanced Science*, 8(12):2100707, June 2021.

[81] Peter Henderson, Jieru Hu, Joshua Romoff, Emma Brunskill, Dan Jurafsky, and Joelle Pineau. Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning. *The Journal of Machine Learning Research*, 21(1):248:10039–248:10081, June 2022.

[82] Daniel D. Garcia-Swartz and Martin Campbell-Kelly. *Cellular: An Economic and Business History of the International Mobile-Phone Industry*. The MIT Press, Cambridge, 2022.

[83] Xingqin Lin and Namyoon Lee. *5G and Beyond: Fundamentals and Standards*. Springer, Cham, Switzerland, March 2021.

[84] Li-Hsiang Shen, Kai-Ten Feng, and Lajos Hanzo. Five Facets of 6G: Research Challenges and Opportunities. *ACM Computing Surveys*, 55(11):1–39, November 2023.

[85] Ericsson. Ericsson Mobility Report November 2022. Technical report, Ericsson, November 2022.

[86] Jinyong Cheon and Ho-Shin Cho. Power Allocation Scheme for Non-Orthogonal Multiple Access in Underwater Acoustic Communications. *Sensors*, 17(11):2465, November 2017.

[87] Zhiqiang Wei, Jinhong Yuan, Derrick Wing Kwan Ng, Maged Elkashlan, and Zhiguo Ding. A Survey of Downlink Non-orthogonal Multiple Access for 5G Wireless Communication Networks. arXiv, September 2016. arXiv:1609.01856 [cs, math].

[88] Stelios Timotheou and Ioannis Krikidis. Fairness for Non-Orthogonal Multiple Access in 5G Systems. *IEEE Signal Processing Letters*, 22(10):1647–1651, October 2015.

[89] Zhiguo Ding, Zheng Yang, Pingzhi Fan, and H. Vincent Poor. On the Performance of Non-Orthogonal Multiple Access in 5G Systems with Randomly Deployed Users. *IEEE Signal Processing Letters*, 21(12):1501–1505, December 2014.

[90] Ningbo Zhang, Jing Wang, Guixia Kang, and Yang Liu. Uplink Nonorthogonal Multiple Access in 5G Systems. *IEEE Communications Letters*, 20(3):458–461, March 2016.

[91] Yuanwei Liu, Zhijin Qin, Maged Elkashlan, Arumugam Nallanathan, and Julie A. McCann. Non-Orthogonal Multiple Access in Large-Scale Heterogeneous Networks. *IEEE Journal on Selected Areas in Communications*, 35(12):2667–2680, December 2017.

[92] Zhiguo Ding, Pingzhi Fan, and H. Vincent Poor. Random Beamforming in Millimeter-Wave NOMA Networks. *IEEE Access*, 5:7667–7681, February 2017.

[93] Jingjing Cui, Yuanwei Liu, Zhiguo Ding, Pingzhi Fan, and Arumugam Nallanathan. Optimal User Scheduling and Power Allocation for Millimeter Wave NOMA Systems. *IEEE Transactions on Wireless Communications*, 17(3):1502–1517, March 2018.

[94] Zhenyu Xiao, Lipeng Zhu, Jinho Choi, Pengfei Xia, and Xiang-Gen Xia. Joint Power Allocation and Beamforming for Non-Orthogonal Multiple Access (NOMA) in 5G Millimeter Wave Communications. *IEEE Transactions on Wireless Communications*, 17(5):2961–2974, May 2018.

[95] Fei Liu, Petri Mähönen, and Marina Petrova. Proportional Fairness-Based User Pairing and Power Allocation for Non-Orthogonal Multiple Access. In *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1127–1131, August 2015.

Part II

Appended Paper Contributions

Appendix A

The GCD and PSR OMA Papers

A.1 A Learning-Automata Based Solution for Non-equal Partitioning: Partitions with Common GCD Sizes

This paper has been published as:

R. O. Omslandseter, L. Jiao, and J. B. Oommen, "A Learning-Automata Based Solution for Non-equal Partitioning: Partitions with Common GCD Sizes," *Advances and Trends in Artificial Intelligence, From Theory to Practice, IEA/AIE 2021*, vol 12799, pp. 227–239, Springer International Publishing, July 2021.
DOI: https://doi.org/10.1007/978-3-030-79463-7_19

A Learning-Automata Based Solution for Non-Equal Partitioning: Partitions with Common GCD Sizes

Rebekka Olsson Omslandseter[1], Lei Jiao[1], and B. John Oommen[1,2]

[1] University of Agder, Grimstad, Norway {rebekka.o.omslandseter,lei.jiao}@uia.no
[2] Carleton University, Ottawa, Canada oommen@scs.carleton.ca

Abstract. The Object Migration Automata (OMA) has been used as a powerful tool to resolve real-life partitioning problems in random Environments. The virgin OMA has also been enhanced by incorporating the latest strategies in Learning Automata (LA), namely the Pursuit and Transitivity phenomena. However, the single major handicap that it possesses is the fact that the number of objects in each partition must be equal. Obviously, one does not always encounter problems with equally-sized groups[3]. This paper is the pioneering attempt to relax this constraint. It proposes a novel solution that tackles partitioning problems where the partition sizes can be both equal and/or *unequal*, but when the cardinalities of the true partitions have a Greatest Common Divisor (GCD). However, on attempting to resolve this less-constrained version, we encounter a few problems that deal with implementing the inter-partition migration of the objects. To mitigate these, we invoke a strategy that has been earlier used in the theory of automata, namely that of mapping the machine's state space onto a larger space. This paper details how this strategy can be incorporated, and how such problems can be solved. In essence, it presents the design, implementation, and testing of a novel OMA-based method that can be implemented with the OMA itself, and also in all of its existing variants, including those incorporating the Pursuit and Transitivity phenomena. Numerical results demonstrate that the new approach can efficiently solve partitioning problems with partitions that have a common GCD.

Keywords: Learning Automata · Object Migration Automata · Object Partitioning with GCD

1 Introduction

Object Partitioning Problems (OPPs): OPPs, where the true data elements are represented as "abstract" objects, concern dividing a set of elements into subsets based on a certain underlying criterion. OPPs are NP-hard and have been studied since the 1970s. Within OPPs, the sub-field of Equi-Partitioning Problems (EPPs) [3], where all the partitions are of equal sizes, have been solved efficiently using Learning Automata (LA). To solve EPPs, LA-based Object Migration Automata (OMA) algorithms, based on the semi-supervised Reinforcement Learning (RL) paradigm, have demonstrated a superior efficiency, when compared with former algorithms [8–11].

[3] When the true underlying problem has non-equally-sized groups, the OMA reports the best equally-sized solution as the recommended partition.

Observe that the nature of the "true" underlying partitioning problem is always unknown. However, the system presents a sequence of queries that are a realization of objects belonging together. The OMA uses this information to infer and converge to the near-optimal groupings. Essentially, OMA-based solutions are clustering algorithms, except that they do not require an imposed distance-based relation between the objects.

Existing OMA Algorithms: There are different types of OMA algorithms, namely the original OMA, the Enhanced OMA (EOMA), the Pursuit EOMA (PEOMA), and the Transitivity PEOMA (TPEOMA)[4]. Of these algorithms, the OMA is the original pioneering solution [3, 4]. Later, an enhancement to the OMA, termed the EOMA, was proposed in [2], and this prevent the so-called *Deadlock Situation*. The authors of [11] and [8] proposed the improved PEOMA, which incorporated the Pursuit concept (already established in the LA literature) into the EOMA, reducing the levels of noise presented to the learning mechanism. Thereafter, the TPEOMA was introduced in [10], where the transitivity phenomenon was further augmented into the PEOMA algorithm, ensuring even better results in certain Environments and reducing the required number of queries before convergence [7]. Numerous applications of OMA-based algorithm have been in reported in different fields, including that of increasing the trustworthiness of reputation systems [12], and user grouping in mobile radio communications [6]. A detailed survey of OMA-based solutions for various applications is included in [7].

Limitations of Existing OMA Solutions: The developments in the field of OMA have considerably improved their respective performances. However, one salient issue remains unresolved, namely the restriction that the algorithms can only handle partitioning problems where the partitions are equally-sized. There are currently no solutions reported in the literature to address this prominent issue.

Relaxing the Limitations of OMA Solutions: We now state the main goal of this research. In this paper, we relax the equi-partitioning constraint needed for the existing OMA algorithms, by introducing the Greatest Common Divisor OMA (GCD-OMA) algorithm. The fascinating aspect of this novel concept is that it can be implemented in *all* of the current OMA variants. Our proposed solution can solve both Non-Equal Partitioning Problems (NEEPs) and EPPs, whenever the partition sizes possess a non-unity GCD between them. For example, the unknown state of nature may be a partitioning problem that has three objects in one group, six in the second and twelve in the third. However, it will not be able to handle partitions that have three objects in one group and thirteen in the second, since the partition sizes do not have a non-unity GCD.

The Paper's Contributions: The contributions of this paper are as follows:

1. We present the novel GCD-OMA algorithm, whose fundamental paradigm can be incorporated in all the reported versions of OMA algorithms.
2. We formalize a new evaluation criterion for assessing the performance of OMA algorithms. This criterion can also be used for evaluating the accuracies of other algorithms that can solve similar partitioning problems.
3. By resorting to a rigorous experimental regime, we demonstrate the efficiency of the algorithms.

[4] It is clearly, impossible to survey all these families in this short paper. Apart from those mentioned below, the Pursuit OMA (POMA) is another version of the OMA. The concepts motivating the POMA are similar to its PEOMA variant, and its details can be found in [9].

The structure of the paper is organized as follows. In Section 2, we formulate the nature of the set of partitioning problems studied in this paper, and analyze their complexities. Then, in Section 3, we present the GCD-OMA algorithm in detail, including its Reward and Penalty modules. The performance of the proposed algorithm is presented in Section 4, after which we conclude the paper in Section 5.

2 Problem Formulation

The partitioning problem is formalized as follows: We are dealing with an Environment containing O objects, where the set of objects is denoted by $O = \{o_1, o_2, ..., o_O\}$. Our goal is to partition these objects into K disjoint partitions, and the given set of partitions is indicated by $\mathcal{K}$, where $\mathcal{K} = \{\rho_1, \rho_2 ..., \rho_K\}$. For example, partition ρ_1 might consist of o_1, o_2 and o_3, denoted as $\rho_1 = \{o_1, o_2, o_3\}$. The problem, however, is that the identities of the objects that should be grouped together are unknown, but are based on a specific but hidden criterion, known only to an "Oracle", referred to as the "State of Nature". The Oracle *noisily* presents the objects that should be together in pairs, where the degree of noise specifies the difficulty of the problem.

We assume that there is an true partitioning of the objects, Δ^*, and the solution algorithm determines a partitioning, say Δ^+. The solution is optimal if $\Delta^+ = \Delta^*$. The initialization of the objects before partitioning starts is indicated by Δ^0.

2.1 Complexity

The complexity of the problems that can be solved using the existing OMA algorithms and the GCD-OMA algorithms is related to their respective combinatorics. We emphasize that, in reality, we cannot perform an exhaustive search to determine the optimal partitioning. This is because, in traditional OMA problems, we are only presented with queries encountered as time proceeds. Unfortunately, we do not have a performance parameter that directly indicates 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[5]. Note that we consider the Bell number to be unordered because we do not care about the order of the objects. Rather, we are only concerned about whether the objects are grouped or not. 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, and the O-th Bell number is given by $B_O = \sum_{k=1}^{O} \left\{ {O \atop k} \right\}$. Here $\left\{ {O \atop k} \right\}$ is the Stirling numbers of the second kind [1], and $k \in \{1, ..., O\}$. For the O-th Bell number, it follows that $\left(\frac{O}{e \ln O}\right)^O < B_O < \left(\frac{O}{e^{1-\lambda} \ln O}\right)^O$, 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 an NEPP. Consequently, what we need to consider is the different combinations of objects in the various partitions.

[5] This is a count of the different partitions that can be established from a set with O elements.

In general, the number of possible combinations for partitioning problems, where the cardinalities are defined, is given by:

$$W = \frac{O!}{(u!)^x x!(v!)^y y!...(w!)^z z!}, \tag{1}$$

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, $ux + vy + ... + wz = O$. When all the groups are of equal size, we have the combination number W as:

$$W = \frac{O!}{\left(\frac{O}{K}!\right)^K K!}, \tag{2}$$

where $\frac{O}{K}$ is an integer, and consequently, such partitioning problems are also characterized by a combinatorial issue. However, this number is significantly smaller than the one given by the Bell numbers.

In addition to the combinatorial complexity of the problem, the interactions between the Environment and the algorithm is also contaminated by noise. In other words, the queries may include misleading messages. Due to the system's stochastic nature, the problem is more complicated than just finding an instantaneous optimal partitioning, because the optimal partitioning is defined *stochastically*.

2.2 Evaluation Criteria

We measure the efficiency of OMA algorithms by counting the required queries presented to the LA before convergence. The larger the number of queries needed, the less efficient is the algorithm. 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. But sometimes, these two indices differ.

For the OMA, the EOMA, and their proposed GCD 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, as in the case of standard LA. We will denote the number of queries received before the LA has reached convergence by the parameter, Ψ. 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, and so, a query will not always result in a response from the Environment. 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 number of queries required from the Query Generator before the automaton has converged, we will utilize the parameter, Ψ_Q. Note that for the OMA and the EOMA variants, $\Psi = \Psi_Q$.

The TPEOMA, similar to the PEOMA, also filters out queries before they are given to the LA. However, in the 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. Besides, we introduce the parameter Ψ_T for counting the artificially-generated queries.

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, defined as:

$$\gamma = \frac{\sum_{\forall i,\forall j,i\neq j} \Gamma_{o_i,o_j}}{\sum_{k=1}^{K} \frac{\eta_k!}{2!(\eta_k-2)!}}, \tag{3}$$

where $i,j \in \{1,2,...,O\}$, $i \neq j$ and $k \in \{1,2,...,K\}$. 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 $\sum_{k=1}^{K} \frac{\eta_k!}{2!(\eta_k-2)!}$ indicates the total number of potentially correct queries. Note that the η_k parameter, where $k \in \{1,...,K\}$, is the number of objects in each partition. To determine γ, we need to check all possible query pairs, observe if the objects in a query are grouped both in Δ^+ and Δ^*, and divide this by the total of possible correct queries. More specifically, we define:

$$\Gamma_{o_i,o_j} = \Gamma_{o_j,o_i} = \begin{cases} 1, & \text{if } o_i \text{ and } o_j \text{ is grouped in } \Delta^* \text{ and } \Delta^+, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Clearly, when $\Delta^+ = \Delta^*$, we have 100% accuracy, which implies an optimal solution.

3 The Proposed GCD-OMA Scheme

3.1 The Novel Paradigm: State Expansion

The technique that we use to solve GCD-related OPPs is by invoking a fine, but established methodology that has been used in the theory of Finite State Machines (FSMs). In order to cite its importance, we mention two domains where it has been applied.

Firstly, when designing FSM Acceptors for Regular Languages, one first creates a Non-Deterministic FSM (NDFSM) by using elementary machines, and by including the operations of Concatenation, Union and Kleene-Star. In this way, one is able to obtain the NDFSM for the entire language. Subsequently to obtain the find the *Deterministic* FSM, one transforms the NDFSM into a deterministic one by increasing the number of states to be the power set of the original machine. In this way one can obtain a Deterministic machine with 2^N states, but that is totally equivalent to the N-state NDFSM.

An analogous technique is also used to create LA with *deterministic* Output Matrices, where the Output Matrix of the original LA is stochastic. Again, one transforms this into an equivalent LA, except that the states of the new machine increases. Every state in the new machine is specified by a *pair* which contains information about the state of the old machine *and* the output generated by the old machine. In this way, the output matrix of the new machine is rendered deterministic. The reader should observe that by expanding the number of states, the complexity of the machine does not change, although the *capability* of the machine changes.

This is exactly what we shall do in our particular case. We shall design new machines associated with a given GCD, and coalesce them to design the overall machine.

3.2 Designing the GCD-OMA

In traditional OMA, we handle pairs of objects and try to bring them together. Thus, when the query objects are in the same partition, they are rewarded. They are penalized when they are in different partitions. By intelligently replacing the object that changes its partition, we ensure that the number of objects in each partition always remains the same. In the proposed GCD scheme, all the partition sizes have a common GCD. In this way, we can link some of the "sub-partitions" together, and consider them as being associated with the same partition in terms of their behaviors when it concerns rewards and penalties. We refer to the proposed algorithm as the GCD-OMA. However, because it can be utilized together with any member of the OMA family, the nomenclature would be GCD-OMA, GCD-EOMA, GCD-POMA, GCD-PEOMA, and GCD-TPEOMA depending on the OMA type, where the latter suffix is the type of OMA involved.

To extend the OMA functionality to handle NEPPs with non-unity GCDs, we need to change two fundamental concepts in the OMA algorithms. Firstly, we need to change the initialization of objects to align with the GCD. Secondly, we need to link the required sub-partitions in the OMA together to fulfill the size requirement of the overall partitions. Observe that 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 final solution reported by the LA. Due to these changes, the new functionality affects many parts of the original OMA structure.

To make the partition links, we need to consider the GCD of the partitions. We will denote the GCD of the partitioning problem by $\Lambda > 1$, which can be trivially obtained. After we have determined Λ, we need to link the partitions together in the LA, and consider them as representing a single entity. When a certain partition size is not equal to Λ, we need to conceptually consider two or more partitions together as being a single overall partition. The number of partitions that need to be considered together for a given partition k is indicated by x_k given by $x_k = \frac{\eta_k}{\Lambda}$, where $x_k = 1$ for a partition size equal to Λ, indicating that this 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:

$$\iota_k = \{\max(\iota_{k-1}) + 1, ..., \max(\iota_{k-1}) + x_k S\}, \ \ \forall k, \tag{5}$$

where the state range $\{a, .., b\}$ indicates that the objects with states within a and b are inside partition k. Note that partition 1 ($\rho_1 = 1$), 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 range of states from the previous partition to make the range of states 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, and that S is the number of states per partition R.

Consider an example with the partitioning sizes of $\eta_1 = 3$, $\eta_2 = 9$ and $\eta_3 = 12$. Additionally, we have four states ($S = 4$) in the sub-partitions of the LA. The states of this example are visualized in Figure 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. (5), 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.

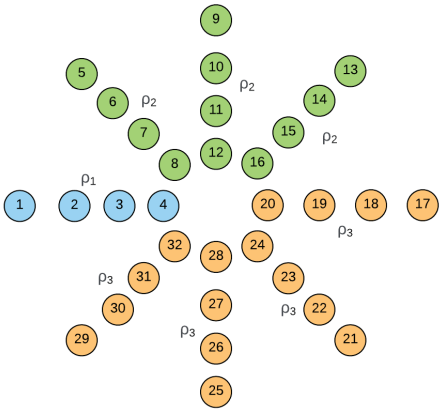


Fig. 1. Example of partition links in GCD with 3 partitions and 4 states as described in the text.

To change the OMA functionality, 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 EOMA version of GCD is described in Algorithm 1. Observe that the GCD-OMA is easily extended to the existing OMA scheme, and is omitted to avoid repetition.

In the GCD schemes, the objects are still initialized in the same manner as before, but instead of placing $\frac{O}{K}$ 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 sub-partitions within the LA. This is done by invoking "EOMA Process Reward" where the objects go deeper into their present action one step at a time, or stay in the same state if they are in the most internal state. Similarly, the objects in a query need to be in different state ranges to be penalized. Again, this is done by invoking "EOMA Process Penalty" where the objects go towards the central boundary states one step at a time, or switch actions when they reach the border.

Algorithm 1 GCD-EOMA

Input:

- The objects $O = \{o_1, ..., o_O\}$.
- S states per sub-partition.
- A sequence of query pairs Υ, where each entry $Q = \{o_i, o_j\}$.
- 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 any of the two most internal states, and the converged partitioning is then 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 $\{1, 2, ..., RS\}$.
- 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 $k \in \{1, 2, ..., K\}$.

1: **while** not converged or $|\Upsilon|$ queries not read **do**
2: Read query $Q = \{o_i, o_j\}$ from Υ
3: **if** θ_i and $\theta_j \in \iota_k$, where $k \in \{1, 2, ..., K\}$ **then** // If the objects are in the same state range
4: EOMA Process Reward
5: **else** // If the objects are in different state ranges
6: EOMA Process Penalty
7: **end if**
8: **end while**
9: Output the final partitioning based on $\theta_i, \forall\, i$. // According to the state ranges

Algorithm 2 EOMA Process Reward

Input:

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

Output:

- The next states of o_i and o_j.

1: **if** $\theta_i \bmod S \neq 1$ **then**
2: $\theta_i = \theta_i - 1$ // Move o_i towards the innermost state
3: **end if**
4: **if** $\theta_j \bmod S \neq 1$ **then**
5: $\theta_j = \theta_j - 1$ // Move o_j towards the innermost state
6: **end if**

4 Experimental Results

In this section[6], we demonstrate the performance of GCD-OMA types for various degrees of noise. Section 4.1 demonstrates results for EPPs compared with other existing

[6] The results presented here are a brief summary of all the results obtained for numerous settings. The detailed set of results are found in the Masters Thesis of the First Author [5].

Algorithm 3 EOMA Process Penalty

Input:
- The query $Q = \{o_i, o_j\}$.
- The states of the objects in Q ($\{\theta_i, \theta_j\}$).

Output:
- The next states of o_i and o_j.

if θ_i mod $S \neq 0$ and θ_j mod $S \neq 0$ **then** // Neither are in boundary
  $\theta_i = \theta_i + 1$
  $\theta_j = \theta_j + 1$
else if θ_i mod $S \neq 0$ and θ_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 (o_i) closest to boundary
  $\theta_j = \theta_i$
  $\theta_l = temp$
else if θ_i mod $S = 0$ and θ_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 (o_j) closest to boundary
  $\theta_i = \theta_j$
  $\theta_l = temp$
else // Both are in boundary states
  $temp = \theta_i$ or θ_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

OMA algorithms. Section 4.2 demonstrates the GCD's performance for NEPPs, which cannot be compared with any of the existing OMA algorithms, as they are unable to handle problems of these kinds. Furthermore, in this context, noise is referred to as queries of objects that are not together in Δ^* but are presented to the LA. A system with noisy queries might also yield a slower convergence rate than a system with fewer (or zero) noisy queries. Consequently, we use:

$$Noise = 1 - \Pi_{o_i,o_j} = 1 - \Pi_{o_j,o_i}, \quad \text{for } o_i, o_j \in \Delta^*, \forall i, j,$$

as the probability reference for LA being presented with a noisy query in the simulations. To clarify, Π_{o_i,o_j} is the probability of o_i and o_j being accessed together and being together in Δ^*. For all the simulations, we utilized 100,000 queries as the maximum number of queries. If the OMA algorithm had not converged within the consideration of $|\Upsilon| = 10^5$, we deemed that the algorithm *had not converged*.

4.1 Existing OMA and GCD-OMA for an EPP

Let us first consider the simulations for an EPP where we simulated a partitioning problem with 30 objects to be partitioned into three partitions, implying that $\frac{O}{K} = 10$. Ta-

ble 1 show simulation results for different existing OMA types, and Table 2 presents results obtained for the GCD-OMA types. GCD-EOMA required approximately 307 and 422 queries before convergence for 0% and 10% noise, respectively. These convergence rate levels are almost equal to those of the existing EOMA algorithm given in Table 1. As the noise level increased, the number of iterations increased. As more noisy queries are presented to the LA, more objects are "misguided" to be together, even if the contrary represents reality. Clearly, the GCD-OMA types and the existing OMA algorithms had similar performance. This behavior is expected. When GCD-OMA types were 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 OMAs.

Note that Ψ indicates the number of queries considered by the LA, Ψ_Q the total number of queries generated, and Ψ_T the queries made from the concept of transitivity in the TPEOMA. For PEOMA, we have to include the parameter κ, indicating the number of queries before we decide to start filtering the queries based on their likeliness before letting the LA process it (pursuit). Additionally, we have the parameter τ, indicating the threshold for whether a query should be considered or not [8, 10].

Type	Noise	γ	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T	κ	τ
EOMA	0%	100%	100%	0%	305.36	305.36	-	-	-
EOMA	10%	100%	100%	0%	425.08	425.08	-	-	-
PEOMA	0%	100%	100%	0%	307.42	309.71	-	270	$\frac{0.1}{O}$
PEOMA	10%	100%	100%	0%	398.11	417.58	-	270	$\frac{0.1}{O}$
TPEOMA	0%	100%	100%	0%	369.55	275.46	96.28	270	$\frac{0.2}{O}$
TPEOMA	10%	100%	100%	0%	555.63	316.91	253.81	270	$\frac{0.2}{O}$

Table 1. Statistics of existing OMA types for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

Type	Noise	γ	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T	κ	τ
GCD-EOMA	0%	100%	100%	0%	307.04	307.04	-	-	-
GCD-EOMA	10%	100%	100%	0%	421.89	421.89	-	-	-
GCD-PEOMA	0%	100%	100%	0%	303.84	305.90	-	270	$\frac{0.1}{O}$
GCD-PEOMA	10%	100%	100%	0%	398.39	417.96	-	270	$\frac{0.1}{O}$
GCD-TPEOMA	0%	100%	100%	0%	371.59	275.37	98.54	270	$\frac{0.2}{O}$
GCD-TPEOMA	10%	100%	100%	0%	553.50	316.94	251.72	270	$\frac{0.2}{O}$

Table 2. Statistics of GCD-OMA types for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

4.2 GCD-OMA variants for NEPPs

This section presents the results for the GCD-OMA types' NEPPs with a non-unity GCD between the respective partition sizes. As demonstrated in Section 4.1, the PEOMA and the TPEOMA variants can enhance the convergence rate of the methods in different ways. The PEOMA is best for systems with higher noise levels, and the TPEOMA is preferred when we have less information (queries) from the system. However, as they are essential 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. We thus present the results only for the GCD-EOMA.

The first problem that we considered had three partitions and 18 objects. The first partition had room for three objects ($\eta_1 = 3$), the second partition had room for six objects ($\eta_2 = 6$), and the last partition had room for nine objects ($\eta_3 = 9$). The second problem that we considered, had 20 objects, where $\eta_1 = 2$, $\eta_2 = 4$, $\eta_3 = 6$ and $\eta_5 = 8$. For this problem, the maximum number of queries was increased to $|\Upsilon| = 10^6$.

Let us first consider the 18-objects case, where the results are listed in Table 3. For 0% noise and three 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 that utilized only three states was because the method achieved convergence only for a minimum of the experiments, with six states.

Observing the results for 10% and 20% noise for GCD-EOMA in Table 3, 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. However, when the system was noise-free or the noise level was lower, the algorithm, astonishingly, performed less accurately, and required more queries if one considered the state depth.

Noise	S	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	$\Psi = \Psi_Q$
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 3. Statistics of GCD-EOMA for the problem with 18 objects ($\eta_1 = 3$, $\eta_2 = 6$, $\eta_3 = 9$) with different noise levels, averaged over 1,000 experiments.

In Table 4, we present the results for the second problem with GCD-EOMA for higher noise levels than for the first problem. The algorithm 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, the noise levels are usually unknown, but they are seldom noise-free.

Noise	Accuracy	$\Delta^+ = \Delta^*$	Not Conv.	$\Psi = \Psi_Q$
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 4. Statistics of GCD-EOMA for the problem with 20 objects ($\eta_1 = 2$, $\eta_2 = 4$, $\eta_3 = 6$, $\eta_4 = 8$), with different noise levels and 6 states, averaged over 1,000 experiments.

From the results, the performance of GCD-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 a desirable phenomenon for the convergence rate, and mitigates problems of having objects "stuck" or locked into a configuration. 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" (or locked in) situations similar, to the Deadlock Situation [2].

5 Conclusions

The existing algorithms within 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. In this paper, we have relaxed the constraint of having equally-sized partitions in OMA schemes. We propose a novel solution that tackles partitioning problems, where the partition sizes can be both equal and/or *unequal*, but when the cardinalities of the true partitions have a GCD. We achieve this by invoking a strategy that has been earlier used in the theory of automata, namely that of mapping the machine's state space onto a larger space. In essence, we have presented the design, implementation, and testing of a novel OMA-based method that can be implemented with the OMA itself, and also in all of its existing variants. The scheme has also been rigorously tested. This paper is a novel contribution and constitutes the first reported OMA-based solution for NEPPs.

References

1. Berend, D., Tassa, T.: Improved Bounds on Bell Numbers and on Moments of Sums of Random Variables. Probability and Mathematical Statistics **30**(2), 185–205 (2010)
2. Gale, W., Das, S., Yu, C.T.: Improvements to an Algorithm for Equipartitioning. IEEE Transactions on Computers **39**(5), 706–710 (May 1990). https://doi.org/10.110912.53585
3. Oommen, B. J., Ma, D.C.Y.: Deterministic Learning Automata Solutions to the Equipartitioning Problem. IEEE Transactions on Computers **37**(1), 2–13 (1988)
4. Oommen, B. J., Ma, D.C.Y.: Stochastic Automata Solutions to the Object Partitioning Problem. The Computer Journal **35**, A105–A120 (1992)
5. Omslandseter, R. O.: Learning Automata-Based Object Partitioning with Pre-Specified Cardinalities. M.S. thesis, University of Agder, Norway (2020)
6. Omslandseter, R.O., Jiao, L., Liu, Y., Oommen, B. J.: User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution. In: Fujita, H., Fournier-Viger, P., Ali, M., Sasaki, J. (eds.) Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices. pp. 299–311. Lecture Notes in Computer Science, Springer International Publishing, Cham (2020). https://doi.org/10.1007978-3-030-55789-8_27
7. Shirvani, A.: Novel Solutions and Applications of the Object Partitioning Problem. Ph.D. thesis, Carleton University, Ottawa (2018)
8. Shirvani, A., Oommen, B. J.: 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). https://doi.org/10.1109ICTCS.2017.40
9. Shirvani, A., Oommen, B. J.: On Enhancing the Object Migration Automaton Using the Pursuit Paradigm. Journal of Computational Science **24**, 329–342 (Jan 2018). https://doi.org/10.1016/j.jocs.2017.08.008, http://www.sciencedirect.com/science/article/pii/S1877750317302259
10. Shirvani, A., Oommen, B. J.: On Invoking Transitivity to Enhance the Pursuit-Oriented Object Migration Automata. IEEE Access **6**, 21668–21681 (2018). https://doi.org/10.1109/ACCESS.2018.2827305
11. Shirvani, A., Oommen, B. J.: On Enhancing the Deadlock-Preventing Object Migration Automaton Using the Pursuit Paradigm. Pattern Analysis and Applications (Apr 2019). https://doi.org/10.1007/s10044-019-00817-z
12. Yazidi, A., Granmo, O.C., Oommen, B. J.: Service Selection in Stochastic Environments: A Learning-Automaton Based Solution. Applied Intelligence **36**(3), 617–637 (2012)

A.2 Object Migration Automata for Non-Equal Partitioning Problems with Known Partition Sizes

This paper has been published as:

R. O. Omslandseter, L. Jiao, and J. B. Oommen, "Object Migration Automata for Non-Equal Partitioning Problems with Known Partition Sizes," *Artificial Intelligence Applications and Innovations, AIAI 2021*, vol 627, pp. 129–142, Springer International Publishing, June 2021.
DOI: https://doi.org/10.1007/978-3-030-79150-6_11

Object Migration Automata for Non-Equal Partitioning Problems with Known Partition Sizes

Rebekka Olsson Omslandseter[1], Lei Jiao[1], and B. John Oommen[1,2]

[1] University of Agder, Grimstad, Norway
[2] Carleton University, Ottawa, Canada
{rebekka.o.omslandseter,lei.jiao}@uia.no, oommen@scs.carleton.ca

Abstract. Solving partitioning problems in random environments is a classic and challenging task, and has numerous applications. The existing Object Migration Automaton (OMA) and its proposed enhancements, which include the Pursuit and Transitivity phenomena, can solve problems with equi-sized partitions. Currently, these solutions also include one where the partition sizes possess a Greatest Common Divisor (GCD). In this paper, we propose an OMA-based solution that can solve problems with both equally and non-equally-sized groups, without restrictions on their sizes. More specifically, our proposed approach, referred to as the Partition Size Required OMA (PSR-OMA), can solve *general* partitioning problems, with the only additional requirement being that the unconstrained partitions' sizes are known *a priori*. The scheme is a fundamental contribution in the field of partitioning algorithms, and the numerical results presented demonstrate that PSR-OMA can solve both equi-partitioning and non-equi-partitioning problems efficiently, and is the only known solution that resolves this problem.

Keywords: Learning Automata · Object Migration Automata · Object Partitioning with Non-Equal Sizes

1 Introduction

What is Object Partitioning: In the Object Partitioning Problem (OPP), we aim to divide a set of "objects" into groups in an optimal manner based on some hidden or unknown criterion. The "object" itself can be the "abstract" representation of a true, real-life data entity. The grouping criterion is always unknown, and only known to an Oracle, which provides information to the system that processes interactions with the real world. A sub-problem and constrained version of the OPP is the Equi-Partitioning Problem (EPP) [5], where all the partitioned groups have an equal size. The family of Object Migration Automata (OMA) algorithms, which are Learning Automata (LA)-based solutions, were first presented in [5, 6]. They could solve EPPs two orders of magnitudes faster than the previously-reported solutions. Over the decades, enhancements have emerged, and include the Enhanced OMA (EOMA) [3], the Pursuit EOMA (PEOMA) [13], and the Transitivity PEOMA (TPEOMA) [12]. In [11], we introduced a solution to the Non-Equal-Partitioning Problem (NEPP) where the sizes of the partitions have a non-unity Greatest Common Divisor (GCD), namely the GCD-OMA.

Although partitioning problems are akin to the related field of clustering, which involves Machine Learning (ML) algorithms like, e.g., K-Means, spectral clustering, and

Gaussian mixtures, it is crucial to understand the distinct aspects of OPPs, and the way by which OMA solve them. While clustering problems, often, have a relation between the objects that can be represented through distance metrics, which in, turn, are required "up-front", OMA algorithms are based on their ability to process *queries* (consisting, for example, of object pairs) presented along time. Consequently, OMA algorithms do not require complete information of the "up-front" inter-relationships between the objects themselves. Thus, OMA algorithms can follow even the stochastic nature of the relations, over time. We emphasize that the true nature of such a partitioning problem is always unknown. However, the presented queries consist of objects that *stochastically* belong together, or should be considered to be together, for some underlying and unknown reasons. The OMA uses this information to infer the groupings.

Applications of Partitioning: One of the numerous applications (extensively given in [9]) for the OMA is cryptanalysis. In [7] and [8], the OMA was employed to solve a cipher using only plaintext and its corresponding ciphertext. This solution achieved a 90% cost reduction compared to its competitors. The authors of [4] proposed an OMA-based scheme to create an image database using conceptually similar images. Recently, the authors of [10] proposed an OMA-based algorithm for mobile radio communications, by partitioning users in a Non-Orthogonal Multiple Access (NOMA) system.

Advancement from the State-of-the-Art: The GCD-OMA represents the state-of-the-art. It rendered the OMA capable of solving NEPPs with *specially-constrained* group cardinalities. However, *this* solution cannot handle general partitioning, due to the GCD requirement on the partition sizes. This paper presents a novel solution, namely the so-called Partition Size Required OMA (PSR-OMA), to EPPs and NEPPs, which does not require a non-unity GCD between the partition sizes. The PSR-OMA can solve partitioning problems with partitions of arbitrary equal or non-equal sizes. The change between the existing OMA solutions and the PSR-OMA is that the latter can adaptively swap the partition sizes. The algorithm still requires us to provide information about the partition sizes, and hence its name, the PSR-OMA. We emphasize that one can use the PSR-OMA with any of the already-existing OMA's "incarnations" and that the PSR-OMA stands apart from the GCD-OMA. The reader should also note that proposing a solution to both EPPs and NEPPs is the same as offering a solution to OPPs, but that we use the EPP and NEPP terminologies to differentiate between the two.

Contributions of this Paper: The contributions of this paper are as follows:

1. We present the novel PSR-OMA scheme applicable for both EPPs and NEPPs, which can be employed with all the existing versions of the OMA algorithms.
2. We demonstrate the convergence and efficiency properties of the PSR-OMAs, showing that it can be used for further applications.

The remainder of the paper is organized as follows. In Section 2, we formulate the nature of the partitioning problems considered in this paper and analyze their complexity. In Section 3, we present the PSR-OMA algorithm in detail. The performance of the proposed algorithm is presented in Section 4, and conclude the paper in Section 5.

2 Problem Formulation

We now formalize the partitioning problem as follows: Our problem consists of O objects, where the set of objects is denoted by $\mathcal{O} = \{o_1, o_2, ..., o_O\}$. We want to divide

the O objects into K disjoint partitions. The set of partitions is indicated by $\mathcal{K}$, where $\mathcal{K} = \{\varrho_1, \varrho_2..., \varrho_K\}$. For example, partition ϱ_3 might consist of o_4, o_5 and o_6, denoted by $\varrho_3 = \{o_4, o_5, o_6\}$. The problem, however, is that the identities of the objects that should be grouped together are unknown, but are based on a specific but hidden criterion, known only to an "Oracle", referred to as the "State of Nature". The Oracle *noisily* presents the objects that should be together in pairs, where the degree of noise specifies the difficulty of the problem. Thus, we assume that there is an true partitioning of the objects, Δ^*, and the solution algorithm determines a partitioning, say Δ^+. The solution is optimal if $\Delta^+ = \Delta^*$. The initialization of the objects is indicated by Δ^0.

The Combinatorics of the OPP: The combinatorial nature of partitioning leads to the complexity of the issues related to the existing OMA and the PSR-OMA algorithms. In OPPs, queries are encountered as time proceeds, and we do not have a performance parameter that directly indicates a particular partitioning's fitness. Thus, we cannot perform an exhaustive search to determine the optimal partitioning of an OPP.

Bell Numbers: An unordered Bell number gives the number of possible partitions of a set of objects. In OPPs, we assume that the ordering of the objects does not matter. Consequently, the Bell number is of an unordered type, and we only consider whether the correct objects are together. Here, we want to partition O objects into K non-empty sets, where each object can only be inside a single group. Accordingly, we have B_O partitioning options, where B_O is the O-th Bell number, and the O-th Bell number is given by $B_O = \sum_{k=1}^{O} \left\{ {O \atop k} \right\}$, with $\left\{ {O \atop k} \right\}$ being the Stirling numbers of the second kind [1], and $k \in \{1, ..., O\}$. The O-th Bell number obeys: $\left(\frac{O}{e \ln O}\right)^O < B_O < \left(\frac{O}{e^{1-\lambda} \ln O}\right)^O$, which has an exponential behavior for O and $\lambda > 0$. However, in our case, the partitioning is pre-defined, independent of whether we have an EPP or an NEPP. Consequently, we need to consider the different combinations of objects in the various partitions. For the partitions, where each of the groups has the possibility to consist of a different number of objects, the number of possible combinations, W, can be expressed as $W = \frac{O!}{\rho_1!\rho_2!\rho_3!...\rho_K!}$, where ρ_k, $k \in \{1, ..., K\}$, is the number of objects in each partition [2]. Further, 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 the given expression for W, 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 in which some of the partition sizes are equal, we have $W = \frac{O!}{(u!)^x x!(v!)^y y!...(w!)^z z!}$, where we have x groups of size u, y groups of size v, and so on for all groups and sizes, implying that, in this case, $ux + vy + ... + wz = O$. Furthermore, when all the groups are of equal size, we can express W as: $W = \frac{O!}{\left(\frac{O}{K}!\right)^K K!}$, where $\frac{O}{K}$ is an integer. As a result of the above, we observe that the solution space for an EPP or an NEPP has a combinatorial complexity.

Complexity of EPPs/NEPPs: EPPs and NEPPs have fewer possible combinations than a Bell number because the partition sizes are specified and known. However, the interactions between the Environment and the algorithm may be contaminated by noise. This means that the queries may include misleading messages. Thus, due to the system's stochastic nature, the problem is more complicated than just finding an optimal partitioning for a given time instant. The optimal partitioning is defined *stochastically*.

Evaluation Criteria: As in [11], γ will be the accuracy of the partitioning determined by the algorithm. We calculate γ by dividing the number of object pairs in Δ^+ that

exist in Δ^* with the total number of possible correct object pairs in Δ^*. Clearly, when $\Delta^+ = \Delta^*$, the scheme will have 100% accuracy, which implies an optimal solution. We denote the number of queries generated from the Environment by Ψ_Q, and let Ψ be the number of queries that the LA has considered. We also use symbol Ψ_T to denote the number of transitivity pairs made in the TPEOMA variant. Note that $\Psi = \Psi_Q$ for the OMA and the EOMA variants.

3 The Proposed PSR-OMA Scheme

The newly-proposed PSR-OMA can handle partitioning problems with partitions of arbitrary non-equal or equal sizes. The primary difference between PSR-OMA and the existing OMA solutions is that PSR-OMA can adaptively swap the partition sizes throughout its operation. In designing it, we encounter some obstacles that are not present for the EPP and the GCD-OMA solution of [11]. Specifically, when we have partitions of pre-specified cardinalities, the objects can become stuck in situations that we refer to as a *Standstill Situations*[3]. Such a "Standstill Situation" is one in which the objects become "stuck" in a loop that might not even be resolved after an infinite time-frame.

Standstill Situation: In this situation, the LA cannot reach convergence due to the constraints imposed by the pre-specified cardinalities. Also, once the partitions have been initialized with their respective number of objects, these allocations will, without modification, be the same. Thus, the objects of a smaller partition, that randomly happen to be within a larger 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. Because the traditional OMA algorithms need 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 associated with solving NEPPs 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 one partition, three objects in the second, and two objects in the third. 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, and we randomly initialize the objects into the different boundary states. After considering an arbitrary number of queries, the EOMA might be stuck in a Standstill Situation, as visualized in Fig. 1.

We observe that in Fig. 1, 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 starting premise is that we specify the cardinalities *a priori*, and make no additional modifications to the algorithm. 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.

[3] The *Standstill Situation* must not be confused by the *Deadlock Situation* previously considered by the authors of [3].

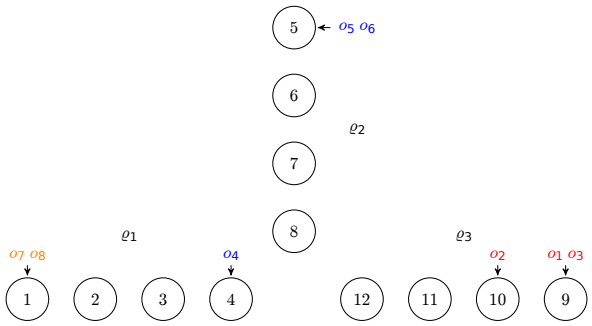


Fig. 1. Example of objects stuck in a Standstill Situation.

The reader should note that the scenario depicted in Fig. 1 is not merely included for explanatory purposes. Rather, this represents an actual Standstill Situation which can occur for many different distributions of objects, and for 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 noise in the system. However, without changing the policy schemes according to the constraints imposed by NEPPs, we can have OMA algorithms that perform poorly by yielding slow convergence, or by not even 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 could resolve the issue, and be able to ultimately converge. However, the convergence rate would be very slow.

Understandably, the Standstill Situation becomes more critical as more partitions are introduced to the OMA algorithm, and its effect 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 larger 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 increases. To mitigate this, the PSR-OMA (which deviates from the OMA) is designed in detail below. In the interest of brevity, the algorithms for the OMA Reward, the OMA Penalty and the EOMA Penalty are not given here. They can be found in [9] and [11], respectively.

Proposed Functionality: The PSR-OMA can be seen to be an extension of the existing OMA algorithms. Its first phase 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 LA's K *boundary* states. 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 pre-specified number of objects in each partition. The new functionality is similar, independent of whether the group sizes are equal or unequal.

Algorithm 1 PSR Process for Standstill Situation

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

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

For ease of explanation, let us assume that o_i is in the innermost state of ϱ_i and o_j is in the boundary state of ϱ_j.

if moving o_j to ϱ_i will let our system keep the specified sizes **then**
  $\theta_j = \theta_i$ // Move o_j to ϱ_i
else // If more than one object is required to fulfill all ρ_k
  for all objects o_x in $\varrho_j \setminus o_j$ **do** // All objects in ϱ_j except o_j
    if $\theta_x = \theta_j$ or $\theta_x = \theta_j - 1$ **then** // If o_x is in (or nearest to) the boundary state
      $I \leftarrow o_x$ // I is the set of possible objects to move
    end if
  end for
  if $|\varrho_i| > |\varrho_j|$ **then** // There are more objects in ϱ_i than in ϱ_j
    $\nu = |\varrho_i| - |\varrho_j|$ // $|\varrho_i|$ is the number of objects in ϱ_i
  else if $|\varrho_i| < |\varrho_j|$ **then** // There are more objects in ϱ_j than in ϱ_i
    $\nu = |\varrho_j| - |\varrho_i|$
  else // This means $|\varrho_i| = |\varrho_j|$
    Continue Process Penalty // Continue with the remaining statements in Alg. 2/3
  end if
  if $|I| + 1 \geq \nu$ **then** // The number of objects in I are bigger than (or equal to) ν
    Randomly select $\nu - 1$ objects from I and put them in a new set J.
    if $|\varrho_i| + \nu$ and $|\varrho_j| - \nu$ fulfills all ρ_k **then** // If the size requirement is fulfilled
      $\theta_j = B_i$ // Move o_j to boundary of ϱ_i
      for all objects o_z in J **do**
        $\theta_z = B_i$ // Move objects in J to boundary state of ϱ_i
      end for
    end if
  else // It was not possible to make a legal swapping of objects
    Continue Process Penalty // Continue with the remaining statements in Alg. 2/3
  end if
end if

In the second phase of the PSR-OMA, we try to mitigate the Standstill Situation by introducing a new policy when the system receives a Penalty. This occurs 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

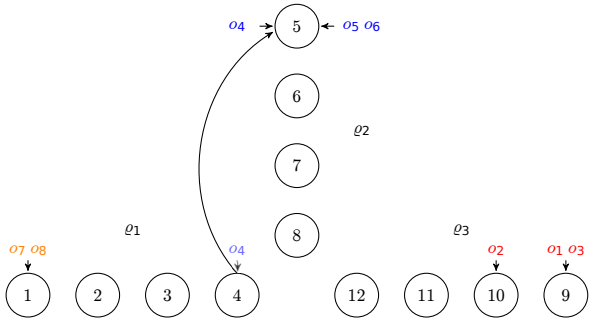


Fig. 2. Example of the Penalty functionality for the Standstill Situation.

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 Fig. 2, where o_4 is allowed to move to the partition of o_5 and o_6, without requiring any replacement.

Migration of Objects: We emphasize 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. 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 1. Observe that for Algorithm 1, we introduce the parameter θ_{B_k}, which indicates 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 not able to swap partition sizes, we handle them as if one object is in the boundary and the other object not being in the boundary according to the EOMA's existing rules.

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, is depicted in Fig. 2.

Implementation Details: The PSR-OMA includes a new initialization of objects. Thus, the objects need to initialized into the partitions according to their pre-specified sizes. This should be done randomly. The second part of the new functionality is invoked as the machine encounters a certain placement of the objects and receives a Penalty. More specifically, the new functionality comes into play when the LA receives a Penalty and one queried object is in the innermost state, and the other queried object is in the boundary state. Consequently, if moving the boundary object, or more objects from the partition of the boundary object, fulfills the size requirements for the partitions,

Algorithm 2 PSR-OMA Process Penalty

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

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

if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S \neq 0$ **then** // Neither are in boundary states
  $\theta_i = \theta_i + 1$
  $\theta_j = \theta_j + 1$
else if $\theta_i \bmod S = 1$ and $\theta_j \bmod S = 0$ **then** // o_i is in innermost state
  PSR Process for Standstill Situation (Algorithm 1)
else if $\theta_i \bmod S = 0$ and $\theta_j \bmod S = 1$ **then** // o_j is in innermost state
  PSR Process for Standstill Situation (Algorithm 1)
else if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S = 0$ **then** // o_j is in boundary state
  $\theta_i = \theta_i + 1$
else if $\theta_i \bmod S = 0$ and $\theta_j \bmod S \neq 0$ **then** // o_i is in boundary state
  $\theta_j = \theta_j + 1$
else // Both are in boundary states
  $temp = \theta_i$ or θ_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

a legal swapping of object(s) from the boundary object's partition to the innermost object's partition is executed. Consequently, the LA is able to change the partition sizes throughout its operation, as long as we, in total, always maintain the pre-specified sizes. By way of example, consider the scenario that we have a problem with the pre-specified sizes of 5, 6 and 7. If ϱ_1 changes from being the size of 5 to 6, the earlier partition with size 6 needs to become the 5-sized one. By operating in this manner, we will always maintain the partition sizes as being 5, 6 and 7.

The reader should observe that the proposed functionality can be directly implemented into the currently-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 2 and Algorithm 3 respectively.

To summarize, for the PSR-OMA its Penalty functionality is given by Algorithm 2, while the rest of the established method remains the same. For the PSR-EOMA, the Penalty scheme is given by Algorithm 3. Again, the other functionalities of the PSR-EOMA behavior are similar to that of the existing EOMA. Additionally, the functionality of "PSR"-based functionalities can be easily extended to the PEOMA and the TPEOMA, yielding what we will refer to as the PSR-PEOMA and PSR-TPEOMA respectively. The details of these LA is trivial and not included to avoid repetition.

Algorithm 3 PSR-EOMA Process Penalty

Input:

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

Output:

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

if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S \neq 0$ **then** // Neither are in boundary states
  $\theta_i = \theta_i + 1$
  $\theta_j = \theta_j + 1$
else if $\theta_i \bmod S = 1$ and $\theta_j \bmod S = 0$ **then** // o_i is in innermost state
  PSR Process for Standstill Situation (Algorithm 1)
else if $\theta_i \bmod S = 0$ and $\theta_j \bmod S = 1$ **then** // o_j is in innermost state
  PSR Process for Standstill Situation (Algorithm 1)
else if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S = 0$ **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 $\theta_i \bmod S = 0$ and $\theta_j \bmod S \neq 0$ **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 θ_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

4 Numerical Results

In this section, we demonstrate the performance of the PSR-OMA, both for an EPP and two NEPPs. Section 4.1 demonstrates results for an EPP, and it is compared with other existing OMA algorithms. Section 4.2 displays the PSR's performance for NEPPs, which cannot be compared with any of the existing OMA algorithms due to their limitation of requiring equally-sized partitions. Our simulations included "Noise", which represents the proportion of queries with objects that did not belong together in Δ^*. Such queries present disinformation to the LA, and indeed, the hardness of the problem (the *Environment*) increases with the level of noise. Therefore, we use:

$$Noise = 1 - Pr\{o_i, o_j \text{ accessed together}\} = 1 - \Pi_{o_i,o_j}, \quad \text{for } o_i, o_j \in \Delta^*, \forall i, j,$$

as the probability measurement for the LA being presented with a noisy query in the simulations [11], to demonstrate its performance in harder Environments. Consequently, Π_{o_i,o_j} is the probability of o_i and o_j being accessed together and being together in Δ^*.

4.1 Existing OMA and PSR-OMA for an EPP

Let us first consider the simulations for an EPP where we simulated a partitioning problem with 30 objects to be grouped into three partitions, implying that $\frac{O}{K} = 10$. Table 1 shows the simulation results for different existing OMA types, and Table 2 presents results obtained for the PSR-OMA types.

One of the main differences between the PSR-EOMA and the existing EOMA is that it considers the scenario 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 is possible without replacement, the new policy does not apply to these problems. We thus expect the PSR-OMA to yield results similar to those of the existing OMA types. The results obtained in Tables 1 and 2 verify this hypothesis, as the existing OMA types and the PSR-OMA types have similar performance for the different noise levels.

Note that for the PEOMA and TPEOMA, we have the κ value indicating the number of queries that have to be processed before we start filtering the queries before letting the LA process them (i.e., deploying the Pursuit concept) and making transitivity pairs. Additionally, we have τ, indicating the threshold for whether a query should be considered or not [12, 13].

Table 1. Statistics of existing OMA types for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

Type	Noise	γ	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T	κ	τ
EOMA	0%	100%	100%	0%	305.36	305.36	-	-	-
EOMA	10%	100%	100%	0%	425.08	425.08	-	-	-
PEOMA	0%	100%	100%	0%	307.42	309.71	-	270	$\frac{0.1}{O}$
PEOMA	10%	100%	100%	0%	398.11	417.58	-	270	$\frac{0.1}{O}$
TPEOMA	0%	100%	100%	0%	369.55	275.46	96.28	270	$\frac{0.2}{O}$
TPEOMA	10%	100%	100%	0%	555.63	316.91	253.81	270	$\frac{0.2}{O}$

Table 2. Statistics of PSR-OMA types for a case involving 30 objects, 3 partitions and 10 states averaged over 1,000 experiments.

Type	Noise	γ	$\Delta^+ = \Delta^*$	Not Conv.	Ψ	Ψ_Q	Ψ_T	κ	τ
PSR-EOMA	0%	100%	100%	0%	304.44	-	-	-	-
PSR-EOMA	10%	100%	100%	0%	417.89	-	-	-	-
PSR-PEOMA	0%	100%	100%	0%	308.65	310.91	-	270	$\frac{0.1}{O}$
PSR-PEOMA	10%	100%	100%	0%	393.14	411.80	-	270	$\frac{0.1}{O}$
PSR-TPEOMA	0%	100%	100%	0%	362.26	274.16	90.08	270	$\frac{0.2}{O}$
PSR-TPEOMA	10%	100%	100%	0%	551.48	315.43	250.13	270	$\frac{0.2}{O}$

4.2 PSR-OMA for NEPPs

We now demonstrate the performance of the PSR-EOMA for general NEPPs. Clearly, the existing OMA types cannot solve these problems because they do not have equally-sized partitions. Further, the reader should observe that unlike the problems presented for the GCD-OMA in [11], these do not possess a non-unity GCD requirement. We configured 10^6 as the maximum number of queries.

We considered two partitioning problems in our simulations. The first problem had "many partitions", and the second problem had "big partition size differences".

These problems are referred to as NEPP 1 and NEPP 2, respectively. The first problem, NEPP 1, has $\rho_1 = 4$, $\rho_1 = 5$, $\rho_2 = 6$, $\rho_3 = 7$, and $\rho_4 = 8$. The second problem, NEPP 2, has $\rho_1 = 4$, $\rho_2 = 9$, and $\rho_3 = 13$. Note that only results for PSR-EOMA are presented here due to space limitations.

Results for the PSR-EOMA for NEPP 1: Let us first consider PSR-EOMA's performance for NEPP 1. In Table 3, the percentage of experiments that discovered the optimal partitioning increases from 91% to 98% and 99% for 10%, 20% and 30% noise, respectively. The PSR-OMA was able to find accurate solutions that were not far from the optimal ones. The accuracy 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 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 and the percentage of finding the optimal partitioning, we understand that for the non-optimal solutions, there were only one or two objects in the incorrect partitioning as the LA converged.

Table 3. Statistics of PSR-EOMA for NEPP 1, with different noise levels and 6 states, averaged over 100 experiments.

Noise	γ	$\Delta^+ = \Delta^*$	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

Results with PSR-EOMA for NEPP 2: In Table 4, we present the statistics for simulations for NEPP 2 with PSR-EOMA. From these results, we see that the method again had better performance in terms of accuracy and convergence as the noise increased. For 30% noise compared with 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 NEPP 2.

Table 4. Statistics of PSR-EOMA for NEPP 2, with different noise levels and 6 states, averaged over 100 experiments.

Noise	γ	$\Delta^+ = \Delta^*$	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

As the results above indicate, the PSR-EOMA struggled for partitioning problems with lower noise levels as the difference between the partition sizes increased, as for NEPP 1. For such cases, the noise helps the algorithm continue "exploring" by keeping objects in the outer states. This happens when all the objects, except some, are correctly placed. In that case, they might be introduced to a noisy query that could help them get "un-stuck". For more manageable problems, like for NEPP 1, the noise has the opposite effect by increasing its number of required queries. Indeed, for problems with smaller differences between the partition sizes, the noise complicated the LA's convergence by misleading it. For both issues, in general, we attained relatively high accuracy levels.

5 Conclusion

Existing algorithms within the OMA paradigm can only solve partitioning problems with partitions of equal sizes or problems with a GCD between the partition sizes. In this paper, we have proposed a solution that can solve NEPPs in general with known partition sizes. Our experimental results show that the proposed algorithm has comparable performance to the existing algorithms regarding solving EPPs and that it can also solve NEPPs accurately. As far as we know, this is the only known solution that resolves this problem.

References

1. Berend, D., Tassa, T.: Improved Bounds on Bell Numbers and on Moments of Sums of Random Variables. Probability and Mathematical Statistics **30**(2), 185–205 (2010)
2. Brualdi, R.A.: Introductory Combinatorics. Pearson, 5th edition edn.
3. Gale, W., Das, S., Yu, C. T.: Improvements to an Algorithm for Equipartitioning. IEEE Transactions on Computers **39**(5), 706–710 (May 1990). https://doi.org/10.1109/12.53585
4. Oommen, B. J., Fothergill, C.: Fast Learning Automaton-Based Image Examination and Retrieval. The Computer Journal **36**(6), 542–553 (1993)
5. Oommen, B. J., Ma, D. C. Y.: Deterministic Learning Automata Solutions to the Equipartitioning Problem. IEEE Transactions on Computers **37**(1), 2–13 (1988)
6. Oommen, B. J., Ma, D. C. Y.: Stochastic Automata Solutions to the Object Partitioning Problem. The Computer Journal **35**, A105–A120 (1992)
7. Oommen, B. J., Zgierski, J. R.: A Learning Automaton Solution to Breaking Substitution Ciphers. IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(2), 185–192 (1993) https://doi.org/10.1109/34.192492
8. Oommen, B. J., Zgierski, J. R.: Breaking Substitution Cyphers Using Stochastic Automata. IEEE Transactions on Pattern Analysis and Machine Intelligence **15**(2), 185–192 (1993) https://doi.org/10.1109/34.192492
9. Omslandseter, R. O.: Learning Automata-Based Object Partitioning with Pre-Specified Cardinalities. 178 (2020), University of Agder,
10. Omslandseter, R. O., Jiao, L., Liu, Y., Oommen, B. J.: User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution. In: IEA/AIE 2020
11. Omslandseter, R. O., Jiao, L., Oommen, B. J.: A Learning-Automata Based Solution for Non-Equal Partitioning: Partitions with Common GCD Sizes. In: IEA/AIE 2021
12. Shirvani, A., Oommen, B. J.: On Invoking Transitivity to Enhance the Pursuit-Oriented Object Migration Automata. IEEE Access **6**, 21668–21681 (2018). https://doi.org/10.1109/ACCESS.2018.2827305
13. Shirvani, A., Oommen, B. J.: On Enhancing the Deadlock-Preventing Object Migration Automaton Using the Pursuit Paradigm. Pattern Analysis and Applications (Apr 2019). https://doi.org/10.1007/s10044-019-00817-z

Appendix B

The HDPA Papers

B.1 The Hierarchical Discrete Learning Automaton Suitable for Environments with Many Actions and High Accuracy Requirements

This paper has been published as:

R. O. Omslandseter, L. Jiao, X. Zhang, A. Yazidi, and J. B. Oommen, "The Hierarchical Discrete Learning Automaton Suitable for Environments with Many Actions and High Accuracy Requirements," *AI 2021: Advances in Artificial Intelligence, AI 2022 (AJCAI 2021)*, vol 13151, pp. 507–518, Springer International Publishing, February 2022.
DOI: https://doi.org/10.1007/978-3-030-97546-3_41

The Hierarchical Discrete Learning Automaton Suitable for Environments with *Many* Actions and *High* Accuracy Requirements

Rebekka Olsson Omslandseter[1], Lei Jiao [1], Xuan Zhang [2], Anis Yazidi [3], and B. John Oommen [1,4]

[1] Dept. of Information and Communication Technology, University of Agder, 4879, Grimstad, Norway, {rebekka.o.omslandseter, lei.jiao}@uia.no
[2] Norwegian Research Centre (NORCE), 4879, Grimstad, Norway
[3] Oslo Metropolitan University, 0167, Oslo, Norway
[4] Carleton University, Ottawa, Canada

Abstract. Since its early beginning, the paradigm of Learning Automata (LA), has attracted much interest. Over the last decades, new concepts and various improvements have been introduced to increase the LA's speed and accuracy, including employing probability updating functions, discretizing the probability space, and implementing the "Pursuit" concept. The concept of incorporating "structure" into the ordering of the LA's actions is one of the latest advancements to the field, leading to the ϵ-optimal Hierarchical Continuous Pursuit LA (HCPA) that has superior performance to other LA variants when the number of actions is *large*. Although the previously proposed HCPA is powerful, its speed has a handicap when the required action probability of an action is approaching unity. The reason for this slow convergence is that the learning parameter operates in a multiplicative manner within the probability space, making the increment of the action probability smaller as its probability becomes close to unity. Therefore, we propose the novel Hierarchical Discrete Learning Automata (HDPA) in this paper, which does not possess the same impediment as the HCPA. The proposed machine infuse the principle of discretization into the action probability vector's updating functionality, where this type of updating is invoked recursively at every depth within a hierarchical tree structure and we pursue the best estimated action in all iterations through utilization of the Estimator phenomenon. The proposed machine is ϵ-optimal, and our experimental results demonstrate that the number of iterations required before convergence is significantly reduced for the HDPA, when compared with the HCPA.

Keywords: Reinforcement Learning · Learning Automata · Hierarchical Discrete Pursuit LA.

1 Introduction

In the field of Learning Automata (LA), non-human agents, implemented through computer programs, find solutions to problems of stochastic nature through the

concept of learning. One of the main advantages of this type of Machine Learning (ML) is the scheme's ability to operate adaptively. The paradigm is based on a learning agent, or Learning Automaton, referred to as a LA, that interacts with a teacher, referred to as the *Environment* [4]. The LA has several actions that often correspond to the different solutions for the given problem. Through selecting actions and getting feedback from the Environment, the LA learns which action (behavior) results in the highest probability of receiving a Reward from the Environment. Although the LA requires feedback from an Environment, it learns in a Semi-Supervised manner. Thus, the LA does not need examples of solutions to learn. They explore different actions and learn via trial-and-error. We can model the Environment in numerous ways. The reader should note that, in this paper, the shortened term LA refers to both the field of Learning Automata and the Learning Automaton according to the context where it appears.

In LA, we evaluate the schemes' performance by the number of iterations needed before convergence and the schemes' accuracy in finding the optimal solution [4]. Improvements that have enhanced the accuracy and speed in LA include using action probability vectors to decide the LA behavior (VSSA), discretizing the probability space of these vectors (Discrete LA), and deploying the "pursuit" concept. In addition, pursuing the most likely action to receive a Reward throughout the LA operation (Estimator-based LA) and ordering the LA in a hierarchical structure (Hierarchical LA) improved the LA's performance further. The structuring of the LA led to the state-of-the-art ϵ-optimal Hierarchical Continuous Pursuit LA (HCPA) [12], which can handle a large number of actions.

Although the HCPA handles a large number of actions, its convergence speed has an impediment when any action probability approaches unity. The reason for this slow convergence is that the learning parameter, and thus the updating of the probabilities operates in a multiplicative manner within the probability space. Consequently, the learning rate decreases as the probability approaches unity. In more detail, as the learning continues, the increment of action probability is less and less, making it more challenging for the LA to converge in the latter phase of learning. This drawback is even more apparent when the criterion for convergence is high, i.e., when high accuracy is required. To solve this problem, we propose the novel ϵ-optimal Hierarchical Discrete Learning Automata (HDPA) scheme in this paper. The HDPA includes all the phenomena mentioned earlier to speed up the convergence when high accuracy is required. The beauty of the HDPA is that the learning speed does not decrease as the learning continues because it operates in a step-wise manner in the probability space. Our simulation results demonstrate that the HDPA has significantly faster convergence for high accuracy requirements. Thus, the HDPA outperforms the state-of-the-art HCPA scheme presented in [12] when the accuracy requirement is high.

Our contributions are summarized as follows:

- We propose the novel HDPA that converges faster than the state-of-the-art HCPA algorithm, when the accuracy requirement is high. The advantages become more pronounced when a large number of actions exist.
- Via extensive simulations, we demonstrate how much the HDPA converged faster than the HCPA for Environments with *many* actions and high accuracy requirements.

2 Related Work

Michael Lvovitch Tsetlin pioneered the field of LA by inventing the Tsetlin Automata (TA) [10], which can be categorized as a Fixed Structure Stochastic Automata (FSSA). The FSSA has a discrete and state-based structure, where the automaton's current state determines its action. The first significant improvement in LA was achieved through the discovery/invention of Variable Structure Stochastic Automata (VSSA), where the action of the LA is selected by randomly sampling an action probability vector. The probability vector is updated through functions according to the Environment's feedback, influencing and changing the behavior of the LA (where this updating functionality can also change over time). While an FSSA needs to move through numerous states before exploring another action, VSSA can possibly explore distinct actions along consecutive iterations, speeding up the exploration of the Environment.

The earlier established variants, the Linear Reward-Penalty (L_{R-P}) scheme, the Linear Reward-Inaction (L_{R-I}) scheme, the Linear Inaction-Penalty (L_{I-P}) scheme, and the Linear Reward-ϵPenalty ($L_{R-\epsilon P}$) scheme in [1] and [4] are all examples of continuous VSSA schemes. The "linear"(L) schemes have such a categorization because the action probabilities are increased in a linear manner. The probabilities of the VSSA can also be increased in a non-linear manner [1, 2, 4]. In mathematical analyses, LA can described through Markov chains, where we have ergodic and absorbing types [1, 8].

The next quantum step in terms of speed and accuracy in LA was achieved through discretizing the action probability space [5]. In traditional VSSA, the action selection probabilities can assume any real value in the interval $[0, 1]$, and the updating is achieved in a multiplicative manner with a learning parameter ($\lambda \in (0, 1)$). The drawback of this continuous approach is its sluggish convergence. As the action probability approach unity, the updating step becomes smaller and smaller, slowing down the algorithm's speed. To address this issue, discretizing the probability space and updating the action probabilities in constant steps was proposed, which significantly improved the convergence speed in LA. The different properties (absorbing and ergodic) of these discretized LA, and their different updating schemes were studied in [5, 6]. In addition, continuous and discretized updating mechanisms with mathematical analyzes are investigated in [7] and [13], respectively.

The invention of Estimator-based Algorithms (EAs) increased the achieved convergence speed of LA even further [9]. The EAs possessed a faster convergence

than all earlier variants. These algorithms are based on VSSA, but in addition, estimates of the reward probabilities of the different actions are maintained in a separate vector. These reward estimates are employed in updating the action selection probabilities, where the LA pursues the currently estimated *best action* in terms of the reward estimates (referred to as the Pursuit paradigm). The reward estimates can be found by Maximum Likelihood, or in a Bayesian manner, investigated in [11]. Thereafter, the researchers combined dicretization and the Pursuit paradigm and introduced the family of Discrete Estimator Algorithms (DEAs) [3].

For the above mentioned LA, the convergence becomes challenging when the number of possible actions is large. Understandably, for VSSA, the action probability vector has a dimension of R (for R actions) and its elements sum up to 1. When R is large, many of the action probabilities can have very small values and may not even be chosen, thus rendering the principle behind VSSA to be void. Inclusion of structure into the field of LA solved this problem, and the HCPA constitutes the state-of-the-art [12]. Although HCPA solves the problem to a certain extent, it is always valuable if we can improve the convergence speed of the algorithm without satisfying the accuracy, leading to the proposed algorithm in this paper.

3 The Proposed Algorithm

The concept of the HDPA is to utilize VSSA, discretizing the probability space, structuring the LA instances in a hierarchical tree structure and incorporating the Estimator concept. In more detail, we organize a set of Discrete Pursuit Automata (DPA) instances in a tree structure, where each instance has a set of actions corresponding to the possible paths down the tree structure from that automaton. The probabilities of these actions are maintained through vectors that are updated in a discretized manner. At the bottom level of the tree, we have the actions that directly interact with the Environment. The HDPA maintains reward estimates of all the actions throughout the tree structure, and we pursue the action with the currently best reward estimate in all iterations according to the pursuit paradigm. The reader should note that, in reality, the reward estimates are only necessary for the actions at the leaf level. We utilize the reward estimates in this way, because the proof of the algorithm's convergence needs the reward estimates along the path[5]. A more detailed explanation of the algorithm is given in what follows.

3.1 The Structure of the HDPA

For ease of explanation of the HDPA in this paper, we utilize 2-action DP_{RI} instances as the LA in the tree structure. The reason for using Reward-Penalty

[5] A formal proof of the HDPA's convergence will be given in an extended version of this paper.

Inaction LA is that they have demonstrated better performance than other configurations [12]. Furthermore, we use 2-actions automatons in our explanations. Therefore, we can model the HDPA as a balanced full binary tree for a problem with 2^K actions, where K is the maximum depth of the tree. The number of actions per LA instance can be changed to another configuration. However, the reader should remember that one of the main reasons for organizing the actions in a tree structure is to mitigate a large action probability vector because of its inferior performance [12]. Therefore, the number of actions in the LA instances should, in any case, be limited. The HDPA in these explanations is, thus, configured to handle 2^K original actions. If the number of original actions is not 2^K, we consider the nearest power of 2 above the number of original actions and configure the excess number of actions with zero Reward probability. To continue our explanations in greater detail, we further formalize the levels in the tree structure as follows:

- **Hierarchical tree structure**: The depth of the tree is indexed by parameter k, $k \in \{0, ..., K\}$. For each level of depth, the number of nodes is indexed by $j \in \{1, ..., 2^k\}$. Note that k and j are also employed to index different LA and actions with their corresponding ranges of definition.
- **The various LA**: The LA $j \in \{1, ..., 2^k\}$ at depth k, is referred to as $\mathcal{A}_{\{k,j\}}$, where $k \in \{0, ..., K-1\}$. The LA at the root is the one at depth 0.
- **The LA at depths from** 0 **to** $K-1$ **(**$0 \leq k < K-1$**)**:
  - Each of the LA, $\mathcal{A}_{\{k,j\}}$, has two actions, denoted by $\alpha_{\{k+1,2j-1\}}$ and $\alpha_{\{k+1,2j\}}$, respectively.
  - Whenever the action $\alpha_{\{k+1,2j-1\}}$ is chosen, the LA, $\mathcal{A}_{\{k+1,2j-1\}}$, at the next level is activated.
  - Whenever the action $\alpha_{\{k+1,2j\}}$ is chosen, the LA, $\mathcal{A}_{\{k+1,2j\}}$, at the next level is activated.
- **The LA at depth** $K-1$ **(**$k = K-1$**)**: The LA at depth $K-1$ select the actual actions to interact with the Environment.
  - All of the LA at depth $K-1$ have two possible actions each, referred to as $\alpha_{\{K,2j-1\}}$ and $\alpha_{\{K,2j\}}$, respectively.
  - The $K-1$ depth of the tree has 2^K actions in total, referred to as $\alpha_{\{K,j\}}$ where $j \in \{1, ..., 2^K\}$.
  - The selected action denoted by: $\alpha_{\{K,j\}}$, is the child of $\mathcal{A}_{\{K-1,\lceil j/2 \rceil\}}$.
- **The actions at level** K **(**$k = K$**)**: At depth K, i.e., at leaves of the tree, we have the actions that directly interact with the Environment.
- $P_{\{k,j\}} = [p_{\{k+1,2j-1\}}, p_{\{k+1,2j\}}]$: The action probability vector of LA $\mathcal{A}_{\{k,j\}}$, where $k \in \{0, ..., K-1\}$ and $j \in \{1, ..., 2^k\}$.

Fig. 1 visualizes the structure of a simple HDPA when four actions exist in the Environment. The leaves of the tree are representations of the actions that the HDPA can take and interact with the Environment. For a HDPA with 2^K actions in the leaf level, the number of LA in the tree structure is $2^K - 1$. In this example, we have three LA, i.e., $\mathcal{A}_{\{0,1\}}$, $\mathcal{A}_{\{1,1\}}$ and $\mathcal{A}_{\{1,2\}}$. As depicted, each LA in the tree has two actions. To choose an action, HDPA follows the

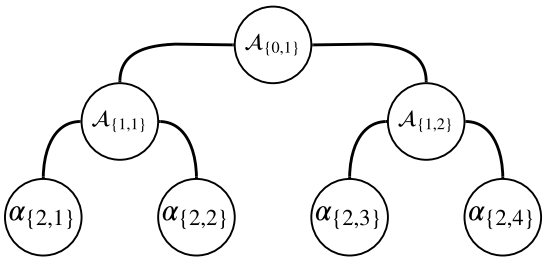


Fig. 1. A visualization of the hierarchy of the HDPA. The instances at the bottom/leaf level of the tree represent the actions that interact with the Environment.

path down the tree by sampling of these action probabilities in the vector. For example, when $\mathcal{A}_{\{0,1\}}$ has an action probability vector of [0.9, 0.1], it selects $\alpha_{\{1,1\}}$ at the root level with probability 0.9. Once $\alpha_{\{1,1\}}$ is chosen, $\mathcal{A}_{\{1,1\}}$ is selected for making the decision at level 1 for the next level (which is at the leaf level in this example), following the action probability vector that $\mathcal{A}_{\{1,1\}}$ maintains. Thereafter, if $\mathcal{A}_{\{1,1\}}$ happens to select the action $\alpha_{\{2,2\}}$, it means that the second action is chosen to interact with the Environment. Thereafter, the HDPA updates its parameters based on the feedback from the Environment. If and only if the Environment offers a Reward, following the pursuit concept, the action probabilities are updated, following the reverse path from the leaf with the current maximum reward estimate to the root. The reader should note that HDPA might reward another action than the one that is currently selected, due the inferior reward estimate in the currently selected action. Independent of whether a Reward or Penalty is received, the reward estimates in all the DPA instances are updated according to whether the action received a Reward or not. The process is detailed as follows:

Parameters:
Δ: The learning parameter, where $0 < \Delta < 1$, and its value is usually configured close to zero.
$u_{\{K,j\}}$: The number of times that action $\alpha_{\{K,j\}}$ was rewarded *when* selected, where $j \in \{1, \ldots, 2^K\}$.
$v_{\{K,j\}}$: The number of times that action $\alpha_{\{K,j\}}$ was selected, where $j \in \{1, \ldots, 2^K\}$.
$\hat{d}_{\{k,j\}}$: The estimated reward probability of action $\alpha_{\{k,j\}}$, $k \in \{1, ..., K\}$, $j \in \{1, \ldots, 2^k\}$. At level K, $\hat{d}_{\{K,j\}}$ is computed as $\hat{d}_{\{K,j\}} = \frac{u_{\{K,j\}}}{v_{\{K,j\}}}$, where $j \in \{1, \ldots, 2^K\}$.
β: The response from the Environment, where $\beta = 0$ corresponds to a Reward, and $\beta = 1$ to a Penalty.
T: Convergence criterion threshold.
We initialize the estimate of the reward probabilities as 0.5, i.e., $u_{\{K,j\}}(0) = 1$, $v_{\{K,j\}}(0) = 2$, thus $\hat{d}_{\{K,j\}}(0) = \frac{1}{2}$. The action probability vector is also initialized as 0.5 for all the LA, i.e., $P_{\{k,j\}}(0) = [\frac{1}{2}, \frac{1}{2}]$, where $k \in \{0, ..., K-1\}$ and $j \in \{1, ..., 2^k\}$.
Begin algorithm:
$t = 0$

Loop

1. **Depths** 0 **to** $K-1$:
   - The LA $\mathcal{A}_{\{0,1\}}$ selects an action by randomly (uniformly) sampling as per its action probability vector $[p_{\{1,1\}}(t), p_{\{1,2\}}(t)]$.
   - Let $j_1(t)$ be the index of the chosen action at depth 0, where $j_1(t) \in \{1, 2\}$.
   - The next LA is activated $\mathcal{A}_{\{1,j_1(t)\}}$ which in turn chooses an action and activates the next LA at depth "2".
   - This process continues including depth $K-1$.
2. **Depth** K:
   - Let $j_K(t)$ be the index of the chosen action at depth K where $j_K(t) \in \{1, \ldots, 2^K\}$.
   - Update $\hat{d}_{\{K,j_K(t)\}}(t)$ based on the response from the Environment at the leaf depth, K:
     $$u_{\{K,j_K(t)\}}(t+1) = u_{\{K,j_K(t)\}}(t) + (1-\beta(t))$$
     $$v_{\{K,j_K(t)\}}(t+1) = v_{\{K,j_K(t)\}}(t) + 1$$
     $$\hat{d}_{\{K,j_K(t)\}}(t+1) = \frac{u_{\{K,j_K(t)\}}(t+1)}{v_{\{K,j_K(t)\}}(t+1)}.$$
   - For all other "leaf actions", where $j \in \{1, ..., 2^K\}$ and $j \neq j_K(t)$:
     $$u_{\{K,j\}}(t+1) = u_{\{K,j\}}(t)$$
     $$v_{\{K,j\}}(t+1) = v_{\{K,j\}}(t)$$
     $$\hat{d}_{\{K,j\}}(t+1) = \frac{u_{\{K,j\}}(t+1)}{v_{\{K,j\}}(t+1)}.$$
3. Define the reward estimate for all other actions along the path to the root, $k \in \{0, ..., K-1\}$ in a recursive manner, where the LA at any one level inherits the feedback from the LA at the level below:
   $$\hat{d}_{\{k,j\}}(t) = \max\left(\hat{d}_{\{k+1,2j-1\}}(t), \hat{d}_{\{k+1,2j\}}(t)\right).$$
4. Proceed to updating the action probability vectors in the LA *along the reverse path from the leaf with the current maximum reward estimate*, as follows:
   - By definition, each LA $j \in \{1, ..., 2^k\}$ at depth k, referred to as $\mathcal{A}_{\{k,j\}}$, where $k \in \{0, ..., K-1\}$, has two actions $\alpha_{\{k+1,2j-1\}}$ and $\alpha_{\{k+1,2j\}}$. Let $j^h_{k+1}(t) \in \{2j-1, 2j\}$ be the index of the larger element between $\hat{d}_{\{k+1,2j-1\}}(t)$ and $\hat{d}_{\{k+1,2j\}}(t)$.
   - Let $\overline{j^h_{k+1}(t)} = \{2j-1, 2j\} \setminus j^h_{k+1}(t)$ be the opposite action, i.e., the one that has the lower reward estimate.
   - Update $p_{\{k+1,j^h_{k+1}(t)\}}$ and $p_{\{k+1,\overline{j^h_{k+1}(t)}\}}$ using the estimates $\hat{d}_{\{k+1,2j-1\}}(t)$ and $\hat{d}_{\{k+1,2j\}}(t)$ as (for all $k \in \{0, ..., K-1\}$):
     If $\beta(t) = 0$ **Then**
     $$p_{\{k+1,j^h_{k+1}(t)\}}(t+1) = \min\left(p_{\{k+1,j^h_{k+1}(t)\}}(t) + \Delta, 1\right),$$
     $$p_{\{k+1,\overline{j^h_{k+1}(t)}\}}(t+1) = 1 - p_{\{k+1,j^h_{k+1}(t)\}}(t+1).$$
     Else
     $$p_{\{k+1,\overline{j^h_{k+1}(t)}\}}(t+1) = p_{\{k+1,\overline{j^h_{k+1}(t)}\}}(t),$$
     $$p_{\{k+1,j^h_{k+1}(t)\}}(t+1) = p_{\{k+1,j^h_{k+1}(t)\}}(t).$$
     EndIf
5. For each $\mathcal{A}_{\{k,j\}}$, if either of its action probabilities $p_{\{k+1,2j-1\}}$ and $p_{\{k+1,2j\}}$ surpasses a threshold T, where T is a positive number that is close to unity, the action probabilities for the HDPA will stop updating, and *convergence* is achieved.
6. $t = t + 1$

EndLoop
End algorithm

4 Experimental Results

To demonstrate the performance of the HDPA compared with the HCPA in [12], we simulated the different schemes' performance for distinct Environments. To enhance the validity of our simulations, we increased the number of experiments and the criteria for convergence compared with the simulations in [12]. As discussed previously, the HCPA has an impediment when the action probability vector approach unity. Our simulations, which we present in more detail shortly, demonstrate the advantage of the HDPA over the HCPA as the convergence criterion is high. The reader should note that we have omitted the comparison with the traditional CPA variants in this paper due to their well-known inferior performance [12].

We conducted experiments for different Environments with *many* actions. The Environments for the 16, 32, and 64 actions were based on the benchmark action probabilities that were first established in [12]. For the 128 actions Environment, we uniformly generated 128 different probabilities between zero and unity, representing the probabilities of the LA receiving a Reward from the Environment. The 128-action Environment's reward probabilities are visualized in Fig. 2.

4.1 The Learning Parameters

From the mathematical proof in [12], and the established theory of VSSA, we understand that when the learning parameter, λ, is sufficiently small, the HCPA will most likely converge to the action that ensures it the maximum probability of obtaining a Reward. The same applies to the value of Δ [14]. In general, a smaller learning parameter results in a slower convergence but has a higher probability of finding the optimal action as its configured value approaches zero. Therefore, tuning the learning parameter is a trade-off between the system's accuracy and convergence speed. In this paper, to find the best value of λ and Δ, we utilized a top-down approach. In more detail, we decreased the value of the learning parameters in a step-wise manner with two decimals precision until their configured value made the LA achieve 100% accuracy for all the given experiments. Consequently, the value of the learning parameters that fulfilled these criteria represents the assumed "best" values of λ and Δ given the distinct Environments used in the simulations.

Although the values for λ and Δ are obtained through extensive testing, it is not entirely certain that one will achieve convergence to the correct action with the found values of λ or Δ for a certain number of consecutive experiments. The Environment and the LA's stochastic behavior over time make it impossible to determine whether the LA will surely converge correctly with a λ or Δ below a certain threshold. The reader should also note that the λ and Δ values are entirely dependent on the Environment's reward probabilities and that the best λ and Δ can vary from case to case. Therefore, we refer to the obtained values of λ and Δ as the "best" learning parameters (and not the *optimal* ones).

4.2 The Average Number of Iterations

In the field of LA, a learning scheme's performance is often measured through the number of iterations required before the algorithm has converged. Due to the stochastic nature of the Environments that LA operates in, we normally measure the average number of iterations, i.e., we conduct many experiments and report the average number

Table 1. HCPA performance for the different simulation Environments.

Number of actions	Mean	Standard deviation (Std.)
16	1,366.61	121.14
32	10,281.84	681.82
64	169,839.67	13,687.48
128	155,088.62	10,613.21

Table 2. HDPA performance for the different simulation Environments.

Number of actions	Mean	Standard deviation (Std.)
16	868.25	135.50
32	6,172.38	744.84
64	100,638.41	17,653.41
128	97,795.59	13,266.12

of iterations required before convergence over a number of experiments. In VSSA, the LA has achieved convergence once the action probability of any one of the actions has reached a certain threshold. The convergence criterion threshold is often configured close to unity. In these simulations, we configured $T = 0.992$, and considered the average of the schemes' performance for 600 experiments.

The simulations discussed in this section, the "best" learning parameters for the Environment with 16 actions were $\lambda = 0.0043$ and $\Delta = 0.0011$. For the Environment with 32 and 64 actions, the best learning parameters were $\lambda = 0.00057$ and $\lambda = 3.6e{-}5$, and $\Delta = 0.00015$ and $\Delta = 9.9e{-}6$, respectively. The "best" obtained values for the 128 action Environment were $\lambda = 3.9e{-}5$ and $\Delta = 9.7e{-}6$. The reader will observe that the learning parameters for the 64 actions and 128 actions cases are quite similar, which is because the 64 actions' Environment's benchmark probabilities were more challenging than the generated Environment for 128 actions.

Tables 1 and 2 show results for our different simulation Environments. In these simulations, we used the benchmark probabilities for the Environments with 16, 32, and 64 actions. For the 128 actions Environment, we used the reward probabilities visualized in Fig. 2. The tables include both algorithms' results and list both the mean and the Standard Deviation (Std). Let us first consider the 16, 32, and 64 actions' Environments, where the HDPA had approximately 37%, 40%, and 41% fewer required iterations compared with the HCPA. Consequently, the benefit of HDPA over HCPA increased with the number of actions. Observing the Std, the HDPA had more variation in the number of iterations for all the three action configurations.

Considering the results obtained for the 128 actions' Environment (based on the reward probabilities visualized in Fig. 2), the HDPA converged within approximately $98,000$ iterations, while the HCPA required $155,000$ iterations before convergence. Comparing the algorithms' Std, the HDPA had more variation in the number of iterations before convergence than the HCPA. Thus, the HDPA was more unpredictable in its number of iterations required before convergence, but needed significantly fewer iterations on average!

Fig. 2. The action probabilities for the 128 actions Environment.

4.3 The Nature of Convergence

The nature of convergence for the HDPA compared with the HCPA is different. As explained earlier, the HDPA updates its action probability in a discretized manner. In contrast, the HCPA updates these probabilities multiplicatively, where the increase in the probability vector can take any value in the interval $[0, 1)$. In Fig. 3 and Fig. 4, we can observe the differences between the operations of the HDPA and the HCPA in greater detail. In these experiments, we increased the convergence criterion to $T = 0.999$ and used the 64 actions Environment from the benchmark probabilities. The figures depict the different schemes' operation for a single iteration. We found the "best" learning parameters through the same top-down approach as explained earlier. The "best" learning parameters were configured as $\Delta = 1e{-}5$ and $\lambda = 5.3e{-}5$, respectively.

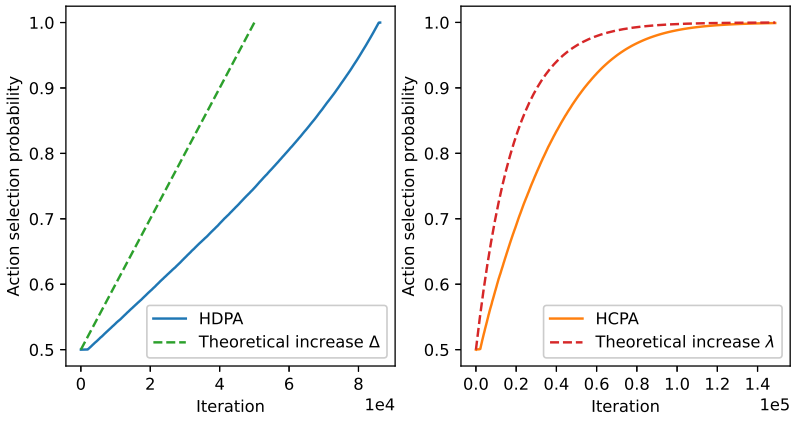


Fig. 3. The action probability of the optimal action per iteration compared to the theoretical increase in the action probability vector for the different schemes.

Fig. 3 depicts the action probability per iteration. As we can observe from the figure, the HDPA requires fewer iterations before convergence than the HCPA does. In addition, we observe the different nature of the two schemes. While the HDPA has a rather linear increase in the action probability, the increase of the action probability for the HCPA scheme slows down as the probability approach unity. Indeed, the HCPA has a faster increase than the HDPA in the initial iterations but suffers from slow convergence because of its behavior as the action probability increases. The reader

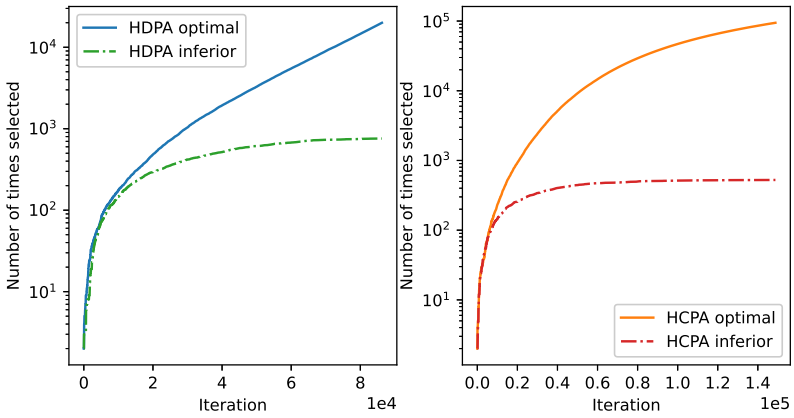


Fig. 4. The number of times that the optimal and the most inferior action were selected for the number of iterations required before convergence for the different schemes.

should also observe that both schemes do not follow the theoretical increase lines in the respective plots. The theoretical increase lines show the convergences when the optimal action is chosen in each iteration, where the action probabilities of the actions in the optimal branch are monotonically increasing. Clearly, these lines depict the theoretical convergence benchmarks without any exploration and are only achievable when the optimal action is known in advance.

In Fig. 4, we observe the difference between the optimal and inferior action for the different schemes. As we observe, the inferior action (the action with the lowest probability of resulting in a Reward) is only selected in the initial phase of the schemes' operation and rarely sampled as the algorithms approach convergence. In contrast, the optimal action is selected more often as the number of iterations increases. As we observe from the figure, the HDPA required significantly fewer iterations than the HCPA. The plots also demonstrate that the inferior action is, indeed, explored throughout the LA operation.

5 Conclusion

In this paper, we proposed the HDPA scheme. The HDPA incorporates all the major phenomena within LA that have improved these algorithms over the last six decades. By implementing VSSA probability updating functionality and discretizing the probability space, utilizing the Estimator phenomenon, and structuring the LA in a hierarchical tree structure akin to the concept of binary search, the HDPA outperforms the state-of-the-art HCPA when the convergence criterion is close to unity, i.e., when the accuracy requirement is high. Our simulations demonstrated the clear advantage of the HDPA over the HCPA for different Environments with many actions. We also demonstrated the difference between the action probability updating of the schemes, and thus, why their performance and convergence speeds are different.

References

1. Lakshmivarahan, S.: Learning Algorithms Theory and Applications. New York Springer-Verlag (1981)
2. Lakshmivarahan, S., Thathachar, M.A.L.: Absolutely expedient algorithms for stochastic automata. IEEE Transactions on Systems, Man, and Cybernetics **3**, 281–286 (1973)
3. Lanctot, J.K., Oommen, B.J.: Discretized estimator learning automata. IEEE Transactions on Systems, Man, and Cybernetics **22**(6), 1473–1483 (1992)
4. Narendra, K.S., Thathachar, M.A.L.: Learning Automata: An Introduction. Courier Corporation (Dec 2012)
5. Oommen, B.J.: Absorbing and ergodic discretized two-action learning automata. IEEE Transactions on Systems, Man, and Cybernetics **16**(2), 282–293 (1986)
6. Oommen, B.J., Christensen, J.P.R.: ϵ-optimal discretized linear reward-penalty learning automata. IEEE Transactions on Systems, Man, and Cybernetics **18**(3), 451–458 (1988)
7. Oommen, B.J., Agache, M.: Continuous and discretized pursuit learning schemes: Various algorithms and their comparison. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **31**(3), 277–287 (2001)
8. Poznyak, A.S., Najim, K.: Learning Automata and Stochastic Optimization, vol. 3. Springer (1997)
9. Thathachar, M.A.L., Sastry, P.S.: Estimator algorithms for learning automata. Proceedings of the Platinum Jubilee Conference on Systems and Signal Processing (1986), Department of Electrical Engineering, Indian Institute of Science
10. Tsetlin, M.L.: Finite automata and the modeling of the simplest forms of behavior. Uspekhi Matem Nauk **8**, 1–26 (1963)
11. X. Zhang, B.J.O., Granmo, O.C.: The design of absorbing bayesian pursuit algorithms and the formal analyses of their ϵ-optimality. Pattern Analysis and Applications **20**, 797–808 (2017)
12. Yazidi, A., Zhang, X., Jiao, L., Oommen, B.J.: The hierarchical continuous pursuit learning automation: A novel scheme for environments with large numbers of actions. IEEE Transactions on Neural Networks and Learning Systems **31**(2), 512–526 (2020)
13. Zhang, X., Granmo, O.C., Oommen, B.J.: Discretized Bayesian pursuit - A new scheme for reinforcement learning. In: Proceedings of IEA-AIE. pp. 784–793. Dalian, China (Jun 2012)
14. Zhang, X., Jiao, L., Oommen, B.J., Granmo, O.C.: A conclusive analysis of the finite-time behavior of the discretized pursuit learning automaton. IEEE transactions on neural networks and learning systems **31**(1), 284–294 (2020)

B.2 The Hierarchical Discrete Pursuit Learning Automaton: A Novel Scheme With Fast Convergence and Epsilon-Optimality

This paper has been published as:

R. O. Omslandseter, L. Jiao, X. Zhang, A. Yazidi, and J. B. Oommen, "The Hierarchical *Discrete* Pursuit Learning Automaton: A Novel Scheme With Fast Convergence and Epsilon-Optimality," *IEEE Transactions on Neural Networks and Learning Systems*, Early Access, pp. 1–15, IEEE, December 2022.
DOI: https://doi.org/10.1109/TNNLS.2022.3226538

The Hierarchical *Discrete* Pursuit Learning Automaton: A Novel Scheme with Fast Convergence and Epsilon-Optimality

Rebekka Olsson Omslandseter, Lei Jiao, *Senior Member, IEEE,* Xuan Zhang, Anis Yazidi, *Senior Member, IEEE,* and B. John Oommen, *Life Fellow, IEEE.*

***Abstract*—Since the early 1960s, the paradigm of Learning Automata (LA) has experienced abundant interest. Arguably, it has also served as the foundation for the phenomenon and field of Reinforcement Learning (RL). Over the decades, new concepts and fundamental *principles* have been introduced to increase the LA's speed and accuracy. These include using probability updating functions, discretizing the probability space, and using the "Pursuit" concept. Very recently, the concept of incorporating "structure" into the ordering of the LA's actions, has improved both the speed and accuracy of the corresponding hierarchical machines, when the number of actions is *large*. This has led to the ϵ-optimal Hierarchical Continuous Pursuit LA (HCPA). This paper[1] pioneers the inclusion of *all* the above-mentioned phenomena into a new single LA, leading to the novel Hierarchical Discretized Pursuit LA (HDPA). Indeed, although the previously-proposed HCPA is powerful, its speed has an impediment when any action probability is close to unity, because the updates of the components of the probability vector are correspondingly smaller when any action probability becomes closer to unity. We propose here, the novel HDPA, where we infuse the phenomenon of discretization into the action probability vector's updating functionality, and which is invoked recursively at every stage of the machine's hierarchical structure. This discretized functionality does not possess the same impediment, because discretization prohibits it. We demonstrate the HDPA's robustness and validity by formally proving the ϵ-optimality by utilizing the moderation property. We also invoke the sub-martingale characteristic at every level, to prove that the action probability of the optimal action converges to unity as time goes to infinity. Apart from the new machine being ϵ-optimal, the numerical results demonstrate that the number of iterations required for convergence is significantly reduced for the HDPA, when compared to the state-of-the-art HCPA scheme.**

***Index Terms*—Reinforcement Learning, Learning Automata, Hierarchical Discrete Pursuit LA, Convergence Analysis**

R. O. Omslandseter and Lei Jiao are with University of Agder, Grimstad, Norway (email: rebekka.o.omslandseter@uia.no, lei.jiao@uia.no).

X. Zhang is with Norwegian Research Center (NORCE), Grimstad, Norway, (email:xuan.z.jiao@gmail.com).

A. Yazidi is with Oslo Metropolitan University, Oslo, Norway (email:anisy@oslomet.no).

B. J. Oommen is a Chancellor's Professor with Carleton University, Ottawa, Canada. He is also an Adjunct Professor with the University of Agder, Grimstad, Norway (email:oommen@scs.carleton.ca). *He dedicates this paper to Neil and Louise Lee, and Michael and Inger-Maria Twilley, who were like parents to his wife and him, when they moved to Canada in 1982.*

[1]We are very grateful to the anonymous Referees of the initial version of this paper. Their comments significantly improved the quality of this final version. A preliminary and very abridged version of some of these results was presented at the 34th Australasian Joint Conference on Artificial Intelligence (AJCAI 2021), in February 2022, in Sydney, Australia.

I. Introduction

The field of Learning Automata (LA), pioneered by Michael Lvovitch Tsetlin in the 1960s [1], has been thoroughly studied over the years [2]. In LA, non-human agents learn with the goal of solving particular tasks through computer programs. Specifically, the concept of LA is based on a *learning agent*, referred to as a LA[2], interacting with a *teacher*, referred to as the Environment. LA entails lightweight adaptive learning schemes that are able to solve complex learning tasks in *stochastic* Environments. The LA learns from the Environment through continuous trial-and-error interactions, and gradually increases its chances of choosing the most favorable action. Without loss of generality, the mapping from the States to the Actions is deterministic.

The LA operates in conjunction with a stochastic Environment, where the LA chooses an action from among a finite set of *actions* offered by the Environment, which, in turn, provides a feedback based on the chosen action. Consequently, the LA adjusts its action based on a selection strategy as per this feedback. Hopefully, this feedback cycle should subsequently lead to the LA making "more intelligent" decisions. The feedback from the Environment is commonly binary, but it can also be from a finite set, or from a continuous range.

There are primarily two categories of LA, namely:

- Fixed Structure Stochastic Automata (FSSA), which have a fixed policy for the inter-state transitions within a finite set of states, where the LA's current state corresponds to its chosen action [2], and where both the updating and decision functionalities are, typically, time-invariant.
- Variable Structure Stochastic Automata (VSSA), where the action selection is based on an action probability vector. In VSSA, updating *functions* are utilized to change the behavior of the LA according to feedbacks from the Environment. These will be explained in more detail in the next section.

LA have scores of applications reported in the Literature. In the interest of brevity, we omit[3] them here and merely include them in the bibliography.

[2]In this paper, the shortened term LA refers to both the field of Learning Automata, the machine, and the Learning Automaton itself, depending on the context in which it appears.

[3]The original submission had a detailed list of the applications of LA from the past decades. Since they are all well cited in the Literature, we merely include them in the bibliography as per the request of the AE and Referees. We are grateful for their input. They can be included if required by the EiC.

A. Goal of this Paper

In any competition, setting an initial record is hard enough[4], but excelling it is a feat. The goal of this paper is to try to attain to a speed/accuracy limit for LA dealing with a large number of actions, that will be hard (if not impossible) to beat!

Improving the speed and accuracy of LA has always involved discovering new concepts and fundamental *principles*. One of the aims of this paper is to record the ways by which the speed/accuracy of LA has been improved over the last six decades. All of these enhancements have incorporated new and fundamental principles that were not invented earlier. Subsequently, combinations of these principles have led to even further improvements. Our goal is to record all of the principles and paradigms, and then combine them efficiently[5].

As briefly explained below, these include using probability updating functions in VSSA, discretizing the probability space, and using the "Pursuit" concept. Very recently, the concept of incorporating "structure" into the ordering of the LA's actions, has improved both the speed and accuracy of the corresponding *hierarchical* machines, when the number of actions is *large*. This has led to the ϵ-optimal Hierarchical Continuous Pursuit LA (HCPA), which is currently the record holder for such Environments. This paper pioneers the inclusion of *all* the above-mentioned phenomena into a new single LA, leading to the novel Hierarchical Discretized Pursuit LA (HDPA).

B. Organization of this paper

The remainder of the article is organized as follows. Section II takes the reader through all the avenues by which the speed/accuracy of LA have been enhanced in quantum jumps or relatively incrementally, over the last six decades. This motivates and sets the stage for the main contribution of this paper, namely, the HDPA. In Section III, we describe, in detail, the new algorithm. In Section IV, we prove the algorithm's convergence property, i.e., its ϵ-optimality. The numerical results are presented in Section V, after which we conclude the paper in the last section.

II. Strategies to Enhance Speed/Accuracy in LA

A. The Infancy: FSSA

As briefly alluded to above, in a stochastic LA, if the state transition probability and output function are constant, i.e., they do not vary with the time step "t" and the input sequence, the automaton is an FSSA. The pioneering and popular examples of these LA were proposed by Tsetlin, Krylov, and Krinsky [1] - all of which are ϵ-optimal under various conditions. These were the primitive ground-breaking LA, and the whole world of LA and RL had their very existence because of them. Their details can be found in [2].

[4]This is apparent from the 100-meter sprint, which is reckoned as the ultimate test of a person's speed. The physical and psychological barrier of completing it in under ten seconds makes the person "a world-class sprinter". Although Carl Lewis pioneered this challenge at 9.97 seconds in 1983, the current record holder is Usain Bolt who ran it in 9.58. It takes a lot more effort, and years of hard work, to even marginally improve a quantifying performance metric, that is almost at the limit of *par excellence*!

[5]We are not aware of any publication which records these details, and we believe that it will be extremely helpful for future researchers.

B. From FSSA to VSSA

The first quantum increase in speed was achieved by the discovery/invention of VSSA. Unlike FSSA, VSSA are the ones in which the state transition probabilities are not fixed. Here, the state transitions or the action probabilities themselves are updated at every time instant using a suitable scheme.

VSSA are an *order of magnitude* faster than FSSA because:

1) VSSA permit an enhanced stochastic exploration of the action probability space, rather than moving through the states of the machine step-by-step, as FSSA do;
2) Unlike FSSA, VSSA permit a "switch" of actions at every step in time, and not merely at the so-called Boundary states;
3) VSSA also provide a far greater flexibility, because they utilize functions to determine the updating, and the number of functions that can be used is limitless;
4) The transition probabilities and the output function vary with time, and the action probabilities are updated on the basis of the input. The action chosen is dependent on the action probability vector, which is, in turn, updated based on the Reward/Penalty input that the LA receives from the Environment.

VSSA are modeled by a discrete-time Markov Process, defined on a suitable set of states. If a probability updating scheme is time invariant, the action probability vector when $t \geq 0$, $\{P(t)\}_{t\geq 0}$, is a discrete-time, homogenous Markov process, and the probability vector at the current time instant $P(t)$, (along with the action at time t, $\alpha(t)$, and the feedback from the Environment at time t, $\beta(t)$) completely determine $P(t+1)$. Hence, each distinct updating scheme, identifies a different type of learning algorithm. For *Continuous Linear* VSSA, the following four learning schemes are extensively studied in the literature: The well-known Linear Reward-Penalty (L_{R-P}) scheme, the Linear Reward-Inaction (L_{R-I}) scheme, the Linear Inaction-Penalty (L_{I-P}) scheme and the Linear Reward-ϵPenalty ($L_{R-\epsilon P}$) scheme, are examples of *linear* VSSA updating rules [2], [3]. As opposed to these, increasing the probabilities of the LA in a non-linear manner has been investigated in [2], [3], [23].

C. From Continuous VSSA to Discretized VSSA

The next paradigm that was invented/discovered to increase the speed/accuracy of LA was that of discretizing the action probability space. The previous VSSA algorithms are *continuous*, i.e., the action probabilities can assume any real value in the interval $[0, 1]$. In such LA, the choice of an action is determined by a Random Number Generator (RNG). In order to increase the speed of convergence of these LA, Thathachar *et al* [24] introduced the family of *discretized* algorithms which pioneered the discretization of the probability space.

Discretized automata can be perceived to be like a hybrid combination of FSSA and VSSA. Discretization is conceptualized by restricting the probability of choosing the actions to only a fixed number of values in the closed interval $[0, 1]$. Thus, the updating of the action probabilities is achieved in steps, rather than in a continuous manner. The different properties (absorbing and ergodic) of these LA, and the updating

schemes of action probabilities for these discretized automata (like their continuous counterparts) were later studied in detail by Oommen *et al* in [24], [25]. Also, similar to the continuous LA paradigm, the discretized versions, the DL_{RI}, DL_{IP}, and DL_{RP} automata have also been reported.

Originally, the assumption was that the RNGs could generate real values with arbitrary precision. In the case of discretized LA, if an action probability is reasonably close to unity, the probability of choosing that action increases *in a single iteration* to unity (when the conditions are appropriate) directly, rather than asymptotically.

The second important advantage of discretization is that it is more practical since RNGs used by continuous VSSA can only *theoretically* be assumed to be *any* value in the interval $[0, 1]$. But machine implementations use *pseudo*-RNGs, where the set of possible values is not infinite in $[0, 1]$, but finite.

Finally, discretization is also important in terms of implementation and representation. Discretized implementations of automata use *integers* for tracking the number of multiples of the learning parameter, $\frac{1}{N} = \Delta$, where N is the so-called *resolution* parameter. This, not only increases the rate of convergence of the algorithm, but also reduces the time, in terms of the clock cycles it takes for the processor to do each iteration of the task, and the memory needed. By virtue of the above, Discretized algorithms are both more time and space efficient than their continuous counterpart algorithms.

D. The Estimator-based Paradigm

The next major quantum jump in the speed/accuracy of LA was by the discovery/invention of Estimator-based Algorithms (EAs). Just as in the case of the family of discretized algorithms Thathachar and Sastry designed a new-class of algorithms, called the *Estimator Algorithms* [4], which at their time possessed a faster rate of convergence than all the previous families. These algorithms, like the previous ones, maintain, and update an action probability vector. However, unlike the previous ones, these algorithms also keep running estimates for each action that is rewarded, using a *reward-estimate vector*, and then use those estimates in the probability updating equations. The reward estimates vector is, typically, denoted in the literature by $\hat{D}(t) = [\hat{d}_1(t), \ldots, \hat{d}_r(t)]^T$. The corresponding state vector is denoted by $Q(t) = \langle P(t), \hat{D}(t) \rangle$, and the estimates can be computed using a Maximum Likelihood scheme (see below), or in a Bayesian manner [26].

The reason for the quantum increase in speed is because in EAs, the convergence involves two intertwined phenomena, namely the convergence of the reward estimates, $\hat{D}(t)$, and the convergence of the action probabilities themselves. The combination of these vectors in the updating rule is intricate, and must be done in a delicately-designed manner. By the law of large numbers, if the actions are sampled "enough number of times", their estimates[6] converge to their true values. The Environment thus influences the probability vector both directly and indirectly, the latter being as a result of the estimation of the reward estimates of the different actions. This may, thus, lead to increases in action probabilities for actions different from the currently-rewarded action. This revolutionary concept changed the entire world of LA, and indeed, even though there is an added computational cost involved in maintaining the reward estimates, these estimator algorithms possess an order of magnitude superior performance than the non-estimator algorithms previously introduced.

Pursuit algorithms are a sub-class of EAs that *pursue* an action that the automaton "currently" perceives to be optimal. The first pursuit algorithm, referred to as the CP_{RP} algorithm due to Thathachar and Sastry, pursues the optimal action on Reward and Penalty. Here, the *currently perceived* "best action" is rewarded, and *its* action probability value is increased with a value directly proportional to its distance to unity, whereas the "less optimal actions" are penalized. The cases of changing the action probabilities *only* on reward and ignoring the penalties lead to the CP_{RI} scheme, also described in [27]. Thathachar *et al* [4] introduced the class of continuous EAs, where one pursues not only the best currently-optimal action[7], and Agache *et al* proposed the so-called Generalized Pursuit LA [27].

E. Merging Estimators-based and Discretized Worlds

The next steps in enhancing the speed/accuracy of LA involved merging the properties of the previously-introduced phenomena. In particular, the researchers piggy-backed on the benefits of dicretization and of the Pursuit paradigm. Utilizing the previously proven capabilities of discretization in improving the speed of convergence of the learning algorithms, Lanctôt and Oommen [28] enhanced the Pursuit algorithm and the "Thathachar and Sastry's Estimator" algorithm [4]. This led to the designing of classes of learning algorithms, referred to in the literature as the Discrete Estimator Algorithms (DEAs) [28], which possessed the so-called *Moderation* and *Monotone* Properties. Agache and Oommen [27] provided a discretized version of their GPA algorithm presented earlier. Their algorithm, called the Discretized Generalized Pursuit Algorithm (DGPA), also essentially generalized the "Thathachar and Sastry's Estimator" algorithm [4].

All of these were further investigated in [29] and [30] respectively, where the earlier flawed proofs of the schemes themselves, were perfected.

F. Incorporating Structure

All of the LA schemes that have been discussed till now assumed that the actions were unordered, which, of course, makes sense since the penalty probabilities are unknown. Indeed, why should one action be preferred above the others? The next major quantum jump in the speed and accuracy of designing LA occurred by incorporating *structure* into the ordering of the actions. This represents the current state-of-the-art, and is particularly pertinent when the number of

[6]The convergence proofs of EAs are far more complex than those of traditional LA. This is because, if the accuracies of the estimates are poor because of inadequate estimation (i.e., if the sub-optimal actions are not sampled "enough number of times"), the convergence accuracy can be diminished. We address this issue later.

[7]The probability updates differ depending in whether the reward estimates are smaller or larger than the estimate of the currently selected action.

actions involved, R, is large. In such scenarios, the learning problem becomes extremely complex, which motivated the authors of [31] to devise a scheme by which small subsets of actions (for example, of cardinality two) were compared, and the result of their comparison was trickled up to avoid dealing with R-action LA and vectors.

Unlike the prior art, in the case of FSSA, one requires S-states for each of the R actions. When the number of actions is large, a LA deals with an $R \cdot S \times R \cdot S$-sized Markov chain, and this adds to the sluggishness of the machine. In the case of VSSA, the action probability vector has a dimension of R and its elements sum up to 1. When R is large, many of the action probabilities can have very small values and may not even be chosen, thus rendering the principle behind VSSA to be void. For the families of pursuit algorithms, the problem still exists because one still utilizes the action probability vector with dimension R, which could be large in this setting.

To make the LA work for a large number of actions, the HCPA was developed [31]. In the hierarchical structure[8] of the HCPA, instead of using one CPA with R actions, they employed multiple CPA, and arranged them in different layers. In this way, the authors avoided having insignificant values in the action probability vector. This endowed the HCPA with the ability to handle the cases when R was very large, which was not even feasible for the traditional CPA to solve.

The quantum jump in speed and accuracy was achieved by merging the phenomena of VSSA and EAs, and doing this in an ordered hierarchical manner, where each LA dealt with a small number of actions. Indeed, the convergence speed of the novel LA proposed in [31] was *many* orders of magnitude faster than any of the other legacy LA. In essence:

1) It incorporated the area of "data structures" into the field of LA, and suggested a novel *hierarchical* LA which uses a tree structure as a part of the learning process;
2) The scheme was based on a multi-level hierarchy composed of two-action CPA at each of the levels, where both the estimation required for an EA, and interaction occurred only at the leaves of the hierarchy;
3) The individual LA performed the learning locally and the result of this was trickled-up in a recursive manner by considering *only* a node and its sibling so as to achieve *global learning*. This also mitigated the problem of having very small action probabilities, since every LA dealt with only two actions.

The HCPA is the state-of-the-art in LA. The goal of this paper is to incorporate all of the above phenomena (VSSA, discretization, the Estimator phenomenon and structure), into our present novel contribution, namely the HDPA. The contributions of this present work are thus summarized as follows:

- We propose a novel HDPA that converges faster than the state-of-the-art HCPA algorithms as the convergence criterion is configured close to unity, e.g., above 0.99. The advantage of the HDPA over the HCPA becomes more obvious when one works with a *large* number of actions;
- We prove, using a formal, rigorous mathematics analysis, the ϵ-optimality of HDPA;
- By resorting to simulation results, we quantify, in detail, how much faster the convergence of the HDPA is when compared to the HCPA. We have also stated, for the first-time, a bound for the number of iterations, which is an avenue for future analytical studies.

[8]A notable prior attempt to devise hierarchical LA is due to Papadimitriou [32]. The difference between what the HCPA and we have done (when compared to the work of [32]), is explained, in detail, in [31].

G. Roadmap for Our HDPA

Although the HCPA can work in the scenario when there are a large number of actions, the novelty of this paper is that we have proposed a viable mechanism by which *its* convergence speed can be improved. By some insight, one observes that HCPA has a relatively sluggish convergence, especially when the required convergence accuracy is high, e.g., above 0.99. The reason behind it is that the changes in the probability vector decrease with the number of iterations. As the learning continues, the increment of the superior action probability is correspondingly decreased, making it more difficult to converge in the later phase of learning. To overcome this, we propose the HDPA to speedup the convergence when high convergence accuracy is required. The beauty of HDPA is that learning speed is not decreasing as the learning continues. This is because we piggy-back the phenomenon of discretization – we incorporate all of the above phenomena, i.e., VSSA, discretization, the Estimator phenomenon and structure!

The newly-proposed discretized learning is shown to be faster than what has been achieved previously in the literature for Environments where the convergence criterion is configured to be more than 99%. The speed of the scheme for such convergence criteria is significantly faster than the HCPA, and the gained efficiency is observed to increase as the number of actions increases. Thus, when we are faced with many actions and we are concerned with the accuracy of convergence, the HDPA outperforms the state-of-the-art HCPA scheme presented in [31] in terms of efficiency, i.e., the number of iterations required before convergence.

We conjecture that the HDPA has thus attained to a speed/accuracy limit for LA dealing with a large number of actions, that will be hard (if not impossible) to beat!

III. Description of the HDPA

The HDPA incorporates all of the phenomena detailed in the introduction. More specifically, we organize a set of DPA instances in a hierarchical tree structure, where all the DPA instances have a set of actions corresponding to the possible paths down the tree structure. We maintain the action probabilities of the respective LAs through vectors that are updated in a discretized manner based on whether an action receives a Reward (or Penalty). At the bottom level of the tree, we have the actions that directly interact with the Environment, and we maintain reward estimates of all the different actions throughout the tree. According to the Pursuit concept, the HDPA pursues the currently estimated best action in all iterations. Thus, the action that currently has the best-estimated reward probability is rewarded upon a Reward, regardless of which action actually triggered the Reward. A more throughout explanation is given in below.

A. The Structure of the HDPA

In the interest of simplicity and clarity, in the following explanations, we will utilize a 2-action DP_{RI} instances as the primitive machine in the construction of the HDPA. Consequently, the hierarchy can be organized as a balanced full binary tree for a problem with 2^K actions[9] and a maximum depth of K. If the number of actions is less than 2^K, one can add dummy actions with zero reward probabilities. Likewise, when we have more than 2^K actions, we need to consider the nearest power of two and then set the action probabilities to zero for the excess number of actions. To incorporate the mathematics established in [31] and for ease of the comparisons to the HCPA scheme, we utilize notations that are similar to those of the latter paper. We further formalize the levels in the hierarchy as follows:

- **The depth index of the tree**: For a tree with the maximum depth K, we employ k to index the depth of the tree, where $k \in \{0, 1, \ldots, K\}$.
- **The various LA**: We denote a specific LA by $\mathcal{A}_{\{k,j\}}$, where k refers to its depth and j represents its particular index in depth, k. More specifically, the LA $j \in \{1, ..., 2^k\}$ at depth k, is referred to as $\mathcal{A}_{\{k,j\}}$, where $k \in \{0, ..., K-1\}$. The LA at the top of the hierarchy is the one at depth 0.
- **The LA at depths from** 0 **to** $K-1$ ($0 \leq k < K-1$):
  - Each of the LA, $\mathcal{A}_{\{k,j\}}$, has two actions, denoted by $\alpha_{\{k+1,2j-1\}}$ and $\alpha_{\{k+1,2j\}}$, respectively.
  - Whenever the action $\alpha_{\{k+1,2j-1\}}$ is chosen, the specific LA $\mathcal{A}_{\{k+1,2j-1\}}$, at the next level is activated.
  - Whenever the action $\alpha_{\{k+1,2j\}}$ is chosen, the specific LA $\mathcal{A}_{\{k+1,2j\}}$, at the next level is activated.
  - $\mathcal{A}_{\{k+1,2j-1\}}$ and $\mathcal{A}_{\{k+1,2j\}}$ are referred to as the *Left Child* and the *Right Child* of its parent ($\mathcal{A}_{\{k,j\}}$), respectively.
- **The LA at depth** $K-1$ ($k = K-1$): The LA at depth $K-1$ select the actual actions to interact with the Environment.
  - All of the LA at depth $K-1$ have two possible actions each, referred to as $\alpha_{\{K,2j-1\}}$ and $\alpha_{\{K,2j\}}$, respectively.
  - The $K-1$ depth has 2^K actions in total, referred to as $\alpha_{\{K,j\}}$ where $j \in \{1, ..., 2^K\}$.
  - The selected action denoted by: $\alpha_{\{K,j\}}$, is the child of $\mathcal{A}_{\{K-1,\lceil j/2 \rceil\}}$.
- **The actions at level** K ($k = K$): At depth K, i.e., at the bottom of the tree, we have the actions that directly interact with the Environment.

B. The Concept of the HDPA

As explained above, the concept of the HDPA is to organize the DPA nodes in a tree structure. Observe that any of the various reported instantiations of the DPA can be utilized at every level. However, we have chosen to utilize the DP_{RI} instances, because the Reward-Inaction scheme has demonstrated a superior performance than the Reward-Penalty types [31].

As depicted in Fig. 1, each node, except the nodes at the bottom level, is the parent of two children, i.e., each node maintains a discretized probability vector with two possible actions, corresponding to its children. The HDPA maintains the original actions through the 2-actions DPA instances at the second bottom level of the tree, i.e., the nodes at depth $K-1$. Consequently, if a problem has $2^8 = 256$ actions, there are 128 nodes at the $K-1$ depth of the tree, each maintaining a 2-action probability vector, i.e., 256 actions in total.

By way of example, let us consider the structure in Fig. 1, where we have eight original actions, i.e., eight leaves. In this case, we have seven LA, i.e., $\mathcal{A}_{\{0,1\}}$, $\mathcal{A}_{\{1,1\}}$, $\mathcal{A}_{\{1,2\}}$, $\mathcal{A}_{\{2,1\}}$, $\mathcal{A}_{\{2,2\}}$, $\mathcal{A}_{\{2,3\}}$ and $\mathcal{A}_{\{2,4\}}$. When the HDPA scenario can be structured as a full binary tree, the number of LA needed is given by $2^K - 1$. Each LA maintains an action probability vector of dimension 2. To choose an action, we follow the path down the tree by sampling the action probabilities in these vectors. For example, when $\mathcal{A}_{\{0,1\}}$ has an action probability vector of [0.9, 0.1], it selects $\alpha_{\{1,1\}}$ at the root, with probability 0.9. Once $\alpha_{\{1,1\}}$ is chosen, $\mathcal{A}_{\{1,1\}}$ is selected for making the decision at depth 1 for the next depth in the tree. Let us assume that $\mathcal{A}_{\{1,1\}}$ happens to select the action $\alpha_{\{2,2\}}$, and the LA $\mathcal{A}_{\{2,2\}}$ is consequently activated. After that, $\mathcal{A}_{\{2,2\}}$ selects the action at the leaf depth as per its action probability vector. If $\alpha_{\{3,4\}}$ happens to be chosen, the fourth original action is selected to interact with the Environment in that iteration. The HDPA consequently updates the probability components in the tree based on the feedback from the Environment. More specifically, the reward estimates are updated in the reverse path of the actions selected in the tree. For the updating of the action selection probabilities, it is based on whether the selected leaf receives a Reward or not. If the selected leaf receives a Reward, following the Pursuit concept, we reward all actions in the tree along the path that leads to the leaf with the current best reward estimate. Note that the leaf with the current best reward estimate may not be the selected leaf in the current iteration due to the fact that we only pursue the action that currently most likely receives a Reward, when averaged over all iterations. The above concepts are formalized below.

Notations and definitions:
To clarify the concepts explained above, we use the following definitions in the description of the algorithm:

- The 2^K actions that interact with the Environment are elements from the set $\{\alpha_{\{K,1\}}, \ldots \alpha_{\{K,2^K\}}\}$. Further, the actions $\{\alpha_{\{K,2j-1\}}, \alpha_{\{K,2j\}}\}$ are the only two actions that can be selected at depth $K-1$, namely $\mathcal{A}_{\{K-1,j\}}$.
- Each LA $j \in \{1, ..., 2^k\}$ at depth k, referred to as $\mathcal{A}_{\{k,j\}}$, where $k \in \{0, ..., K-1\}$ has two actions, namely, $\alpha_{\{k+1,2j-1\}}$ and $\alpha_{\{k+1,2j\}}$.
- $P_{\{k,j\}} = [p_{\{k+1,2j-1\}}, p_{\{k+1,2j\}}]$, is the action probability vector of LA $\mathcal{A}_{\{k,j\}}$, where $k \in \{0, ..., K-1\}$ and $j \in \{1, ..., 2^k\}$.

Parameters:

[9]The LA instances can easily be extended to have more actions, but the reader should remember that we endeavor to mitigate the problem of slow convergence associated with *many* actions in the action probability vector. Consequently, the number of actions should, in any case, be limited in consideration of the convergence rate.

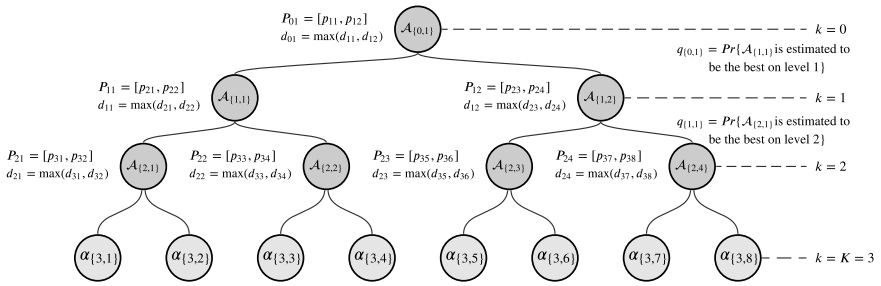


Fig. 1. A visualization of the tree structure in the HDPA with notations as utilized for the explanations in the paper.

Δ: The learning parameter, where $0 < \Delta < 1$, and its value is configured arbitrarily close to zero.
$u_{\{K,j\}}$: The number of times that action $\alpha_{\{K,j\}}$ was rewarded *when* selected, where $j \in \{1, \ldots, 2^K\}$.
$v_{\{K,j\}}$: The number of times that action $\alpha_{\{K,j\}}$ was selected, where $j \in \{1, \ldots, 2^K\}$.
$\hat{d}_{\{k,j\}}$: The estimated reward probability of action $\alpha_{\{k,j\}}$, $k \in \{1, ..., K\}$, $j \in \{1, \ldots, 2^k\}$. At level K, $\hat{d}_{\{K,j\}}$ is computed as $\hat{d}_{\{K,j\}} = \frac{u_{\{K,j\}}}{v_{\{K,j\}}}$, where $j \in \{1, \ldots, 2^K\}$.
β: The response from the Environment, where $\beta = 0$ corresponds to a Reward, and $\beta = 1$ to a Penalty.
T: The convergence criterion threshold.
We initialize the estimates of the reward probabilities as 0.5. Thus, both actions in all the LA in the tree have an initial estimated reward probability of 0.5, i.e., $u_{\{K,j\}}(0) = 1$, $v_{\{K,j\}}(0) = 2$, thus $\hat{d}_{\{K,j\}}(0) = \frac{1}{2}$. The action probability vector is also initialized as 0.5 for all the LA, i.e., $P_{\{k,j\}}(0) = [\frac{1}{2}, \frac{1}{2}]$, where $k \in \{0, ..., K-1\}$ and $j \in \{1, ..., 2^k\}$.

Begin algorithm:
$t = 0$
Loop

1) **Depths** 0 **to** $K-1$:
   - The LA $\mathcal{A}_{\{0,1\}}$ selects an action by randomly (uniformly) sampling as per its action probability vector $[p_{\{1,1\}}(t), p_{\{1,2\}}(t)]$.
   - Let $j_1(t)$ be the index of the chosen action at depth 0, where $j_1(t) \in \{1, 2\}$.
   - The next LA activated is $\mathcal{A}_{\{1,j_1(t)\}}$, in turn, chooses an action and activates the next LA at depth "2".
   - This process continues including depth $K-1$.
2) **Depth** K:
   - Let $j_K(t)$ be the index of the action chosen at depth K, where $j_K(t) \in \{1, \ldots, 2^K\}$.
   - Update $\hat{d}_{\{K,j_K(t)\}}(t)$ based on the response from the Environment at the leaf depth, K:
     $$u_{\{K,j_K(t)\}}(t+1) = u_{\{K,j_K(t)\}}(t) + (1 - \beta(t))$$
     $$v_{\{K,j_K(t)\}}(t+1) = v_{\{K,j_K(t)\}}(t) + 1$$
     $$\hat{d}_{\{K,j_K(t)\}}(t+1) = \frac{u_{\{K,j_K(t)\}}(t+1)}{v_{\{K,j_K(t)\}}(t+1)}.$$
   - For all other "leaf actions", where $j \in \{1, ..., 2^K\}$ and $j \neq j_K(t)$:
     $$u_{\{K,j\}}(t+1) = u_{\{K,j\}}(t)$$
     $$v_{\{K,j\}}(t+1) = v_{\{K,j\}}(t)$$
     $$\hat{d}_{\{K,j\}}(t+1) = \frac{u_{\{K,j\}}(t+1)}{v_{\{K,j\}}(t+1)}.$$
3) Define the reward estimates for all other actions along the path to the root, $k \in \{0, ..., K-1\}$ in a recursive manner, where the LA at any one level inherits the feedback from the LA at the level below as:
   $$\hat{d}_{\{k,j\}}(t) = \max\left(\hat{d}_{\{k+1,2j-1\}}(t), \hat{d}_{\{k+1,2j\}}(t)\right).$$
4) Proceed to update the action probability vectors along the path leading to the leaf with the current maximum reward estimate, as follows:
   - By definition, each LA $j \in \{1, ..., 2^k\}$ at depth k, referred to as $\mathcal{A}_{\{k,j\}}$, where $k \in \{0, ..., K-1\}$, has two actions $\alpha_{\{k+1,2j-1\}}$ and $\alpha_{\{k+1,2j\}}$. Let $j^h_{k+1}(t) \in \{2j-1, 2j\}$ be the index of the larger element between $\hat{d}_{\{k+1,2j-1\}}(t)$ and $\hat{d}_{\{k+1,2j\}}(t)$.
   - Let $\overline{j^h_{k+1}(t)} = \{2j-1, 2j\} \setminus j^h_{k+1}(t)$ be the opposite action, i.e., the one with the lower reward estimate.
   - For all $k \in \{0, ..., K-1\}$, update $p_{\{k+1,j^h_{k+1}(t)\}}$ and $p_{\{k+1,\overline{j^h_{k+1}(t)}\}}$ using the estimates $\hat{d}_{\{k+1,2j-1\}}(t)$ and $\hat{d}_{\{k+1,2j\}}(t)$ as:
     If $\beta(t) = 0$ **Then**
     $$p_{\{k+1,j^h_{k+1}(t)\}}(t+1) = \min\left(p_{\{k+1,j^h_{k+1}(t)\}}(t) + \Delta, 1\right),$$
     $$p_{\{k+1,\overline{j^h_{k+1}(t)}\}}(t+1) = 1 - p_{\{k+1,j^h_{k+1}(t)\}}(t+1).$$
     Else
     $$p_{\{k+1,\overline{j^h_{k+1}(t)}\}}(t+1) = p_{\{k+1,\overline{j^h_{k+1}(t)}\}}(t),$$
     $$p_{\{k+1,j^h_{k+1}(t)\}}(t+1) = p_{\{k+1,j^h_{k+1}(t)\}}(t).$$
     EndIf
5) For each $\mathcal{A}_{\{k,j\}}$, if either of its action probabilities $p_{\{k+1,2j-1\}}$ and $p_{\{k+1,2j\}}$ surpasses a threshold T, where T is a positive number that is close to unity, the action probabilities for the HDPA will stop updating, and *convergence* is achieved.
6) $t = t + 1$

EndLoop
End algorithm

The above algorithm can be simplified because it is unnecessary to update the reward estimates along the path for the algorithm to run. In other words, we only need the estimated reward probabilities for the actions that directly interact with the Environment, i.e., for the leaves. But we have chosen to describe the scheme using the above algorithm, because we utilize, in the next section, the reward estimates along the path in the convergence analysis. In the interest of space

and brevity, as one can see, the detailed description of the *updates* are omitted here, as recommended by the Referees. The interested reader can find them in [40].

IV. Proof of ϵ-Optimality

The proof follows the four-step method established in [33] for the DPA. But in contrast, we prove here that convergence will also occur when the learning units are structured hierarchically. In particular, we consider the moderation property and prove that we have a marginality property along the optimal path. After that, we utilize the sub-martingale property in a level-by-level approach. We finally prove that the probability of the optimal action approaches unity as time goes to infinity.

A. The Moderation Property

We first need to consider the moderation property, proving that under the HDPA, by utilizing a sufficiently small value of the learning parameter, Δ, each action will be selected an arbitrarily large number of times.

Theorem 1. *For any value of $\delta \in (0,1]$ and integer $M < \infty$, there exist a non-zero positive learning parameter, $\Delta_0 < 1$, such that for all $\Delta < \Delta_0$:*

$$Pr\{x\} > 1 - \delta,$$

where x indicates the event that each action is selected more than M times before time t_0.

Proof. The details of the proof that Theorem 1 is true for the DPA can be found in [4], [29], and [34]. We now further elaborate on why this is also true for the HDPA.

In the case of the HDPA, we have 2^K actions at depth K denoted as $\alpha_{\{K,j\}}$ and $j \in \{1, ..., 2^K\}$. Let $Y^t_{\{K,j\}}$ be the number of times action $\alpha_{\{K,j\}}$ is chosen up to time t. To prove Theorem 1, we want to show $Pr\{Y^t_{\{K,j\}} > M\} > 1 - \delta$, which is the same as $Pr\{Y^t_{\{K,j\}} \leq M\} \leq \delta$. The events $\{Y^t_{\{K,j\}} = l\}$ and $\{Y^t_{\{K,j\}} = n\}$ are mutually exclusive for $l \neq n$. Consequently, it follows that:

$$Pr\left\{Y^t_{\{K,j\}} \leq M\right\} = \sum_{l=1}^{M} Pr\left\{Y^t_{\{K,j\}} = l\right\}.$$

Further, the probability of the actions at depth K being chosen is connected to the probabilities in shallower depths as follows:

$$Pr\{\alpha_{\{K,j_K\}} \text{is chosen}\} = p_{\{K,j_K\}} p_{\{K-1,j_{K-1}\}} \cdots p_{\{0,j_0\}},$$

where $j_{K-1} = \lceil j_K/2 \rceil$, $j_{K-2} = \lceil j_{K-1}/2 \rceil$, ..., and $j_0 = \lceil j_1/2 \rceil$. Considering the time aspect, it follows that:

$$Pr\{\alpha_{\{K,j_K\}} \text{is chosen at time } t\} = p_{\{K,j_K\}}(t) p_{\{K-1,j_{K-1}\}}(t) \cdots p_{\{1,j_1\}}(t).$$

Let us further assume that all the LA instances have the same action probability at beginning, i.e., $p_{k,j}(0) = 1/2$ at $t = 0$. For ease of expression, we use $p(0)$ to represent the initial action probabilities, $p_{k,j}(0)$, for all actions. By the *modus operandus* of the various instances of the DPA, we know that the magnitude by which any action probability can decrease in any single iteration is bounded by Δ such that we have:

$$Pr\{\alpha_{\{K,j_K\}} \text{is chosen at time } t\} \leq 1 \text{ and}$$
$$Pr\{\alpha_{\{K,j_K\}} \text{is not chosen at time } t\} \leq (1 - (p(0) - t\Delta)^K).$$

Consider the first t iterations. Using these upper bounds, the probability that $\alpha_{\{K,j_K\}}$ is chosen at most M times among the t iterations has the following upper bound:

$$Pr\{Y^t_{\{K,j\}} \leq M\} = \sum_{l=1}^{M} Pr\{Y^t_{\{K,j\}} = l\} \leq \sum_{l=1}^{M} \binom{t}{l} (1)^l \Psi^{t-l},$$

where $\Psi = 1 - (p(0) - t\Delta)^K$. To show that a sum of M terms is less than δ, it is sufficient to make each element of the sum is less than $\frac{\delta}{M}$. Let us consider the case of $l = m$, where the m-th term times M should be less than δ. Consequently, we need to show that $M\binom{t}{l}(1)^m(1 - (p(0) - t\Delta)^K)^{t-m}$ is bounded by δ. We see that $\binom{t}{l} \leq t^m$, and we need to show that $Mt^m\Psi^{t-m} \leq \delta$. In order to achieve this, $(1 - (p(0) - t\Delta)^K)$ must be strictly less than unity. In order for $(1-(p(0)-t\Delta)^K)$ to be less than unity, we must have $p(0) - t\Delta > 0$, which leads to the bound of Δ, i.e., $\Delta < \frac{p(0)}{t} = \frac{1}{2t}$. Hence, $(1 - (p(0) - t\Delta)^K) < 1$. By definition, $0 < \Delta$ and thus $0 < \Delta < \frac{1}{2t}$. With this value of Δ, we can simplify the analysis, and we now have $Pr\{Y^t_{\{K,j\}} \leq M\} < Mt^m\Psi^{t-m}$, where $0 < \Psi < 1$. We now evaluate the case when $t \to \infty$ as:

$$\lim_{t\to\infty} Mt^m\Psi^{t-m} = M \lim_{t\to\infty} \frac{t^m}{(1/\Psi)^{t-m}}.$$

By using L'Hospital's Rule, it follows that:

$$M \lim_{t\to\infty} \frac{t^m}{(1/\Psi)^{t-m}} = M \lim_{t\to\infty} \frac{m!}{(\ln(1/\psi))^m (1/\Psi)^{t-m}} = 0.$$

Therefore, for every leaf action $\alpha_{\{K,j\}}$, there exists $t = t(j)$ such that $Pr(Y^t_{\{K,j\}} \leq M) \leq \delta$. Since $t > t(j)$ then $Y^{t(j)}_{\{K,j\}} \geq M$ gives $Y^t_{\{K,j\}} \geq M$, and therefore $Pr\{Y^t_{\{K,j\}} \geq M\} \geq Pr\{Y^{t(j)}_{\{K,j\}} \geq M\}$. Consequently, $Pr\{Y^t_{\{K,j\}} \leq M\} \leq \delta$ for all $t > t(j)$. This implies that the probability of an action not being chosen as $t \to \infty$, given the restriction of Δ, is zero.

To complete the proof, let $t_0 = \max_{1\leq j\leq 2^K}\{t(j)\}$. Then for all $t > t_0$ and for all j such that $1 \leq j \leq 2^K$, we have $Pr\{Y^t_{\{K,j\}} \leq M\} \leq \delta$. Theorem 1 is thus proven. □

B. The Marginality Property

For the second part of the proof, we need to show that, given that each action $\alpha_{\{K,j\}}$ is selected a sufficiently large number of times, the reward estimate of the optimal action will remain the largest with sufficiently large probability along the optimal path throughout the hierarchical tree structure. We first establish a baseline for Theorem 2. Let $q_{\{k,j^*_k\}}$ be the probability that the reward estimate, $\hat{d}_{\{K,j^*_K\}}$, of the optimal action, is the largest among all actions of the tree at $\mathcal{A}_{\{k,j^*_k\}}$, where the $*$ is used to indicate the action that corresponds

to the path of the optimal action. This relates to the various depths of the HDPA, when $K > 3$, as follows:

- **The first level (root):** At this depth, we have a single LA $\mathcal{A}_{\{0,1\}}$, and $q_{\{0,1\}}$ is the probability that $\hat{d}_{\{K,j_K^*\}}$ is the maximum among all the 2^K actions.
- **The second level:** $q_{\{1,j_1^*\}}$ is the probability that $\hat{d}_{\{K,j_K^*\}}$ is the maximum among all actions of the tree rooted from $A_{\{1,j_1^*\}}$. There are $2^{(K-1)}$ actions that compete for having the best reward estimate at this depth.
- **The interior level(s):** For the interior depths of the hierarchy, this phenomenon holds as we follow the path down the tree at every level, having fewer actions competing for the best reward estimate as we go further down the tree.
- **The last/leaf level:** $q_{\{K-1,j_{K-1}^*\}}$ is the probability that $\hat{d}_{\{K,j_K^*\}}$ is maximum among the two actions of LA $\mathcal{A}_{\{K-1,j_{K-1}^*\}}$ at this level. There are exactly two actions that compete for having the best reward estimate here.

Theorem 2. *Given a value of $\Delta \in (0,1)$, there exists a time instant denoted by $t_0 < \infty$, such that:*

$$q_{\{k,j_k^*\}} > 1 - \delta,$$

which is true $\forall t > t_0$ and $\forall k \in \{0,1,...,K-1\}$.

Proof. To prove Theorem 2, we use the aspect that $q_{\{0,0\}} < q_{\{1,j_1^*\}} < ... < q_{\{K-1,j_{K-1}^*\}}$, since the probability of being the best from a set of actions is less than the probability of being the best from a subset of actions. Consequently, proving Theorem 2 can be achieved by proving that $q_{\{0,0\}} > 1 - \delta$. Given that Theorem 1 is proven, the proof of Theorem 2 become identical to the corresponding proof for the DPA given in [4], [29], and [34], respectively. The additional details of the proof, are thus, omitted here, to avoid repetition. □

C. The Sub-Martingale Property

In order to conclude the proof in Section IV-C, we first need to show that the HDPA has the sub-martingale property. To do this, we now show that after a time instant t_0, the probability of choosing the optimal action is increasing, *in the expected sense*. This feature is different from the probability of being monotonically increasing, which is a very strong condition.

Theorem 3. *Under the HDPA, the quantity:*

$$\left\{p_{\{k,j_k^*\}}(t)\right\},$$

where $k \in \{1,...,K\}$ and $t > t_0$ is a sub-martingale.

Proof. We first formalize the sub-martingale property by denoting a sequence of random variables as $X_1, X_2, ..., X_t$. The sequence is a sub-martingale if for any time instant t:

$$E[X_t] < \infty \text{ and } E[X_{t+1}|X_1, X_2, ..., X_t] \geq X_t.$$

To prove Theorem 3, we first observe that $p_{\{k,j_k^*\}}$ is a probability, which means $p_{\{k,j_k^*\}} \leq 1 < \infty$. Secondly, we explicitly calculate $E\left[p_{\{k,j_k^*\}}(t)\right]$, for all $k \in \{1,...,K\}$. The proof of this theorem is achieved by an inductive argument. At every level, we consider the nodes that are one and two levels below it, respectively. Indeed, this is true because that is the structure that effectively remains at the specific node, and is essentially depicted in Fig. 1, where the root node of the subtree is determined by the decision of two children and four grandchildren. Once the proof has been proven for such a scenario, the overall proof follows trivially, because the decision of every node is based on the decision of its immediate *two* children and *four* grandchildren. A straightforward inductive argument formalizes this to be true for the overall tree, since it is true at every level for the subtree of depth two from the root of the corresponding subtree. Thus, in what follows, we merely use the tree structure of Fig. 1 to formalize the proof.

We look at the first three levels of the LA hierarchy, and calculate $E\left[p_{\{k,j_k^*\}}(t)\right]$, when $k = 2$. Without loss of generality, we simplify the notations to what are shown in Fig. 1, and let $\mathcal{A}_{\{1,1\}}$ ($\alpha_{\{1,1\}}$) and $\mathcal{A}_{\{2,1\}}$ ($\alpha_{\{2,1\}}$) be the LA (actions) on the optimal path. One can see that this implies that the optimal path is the one consisting of all the left-most nodes in both levels. We denote the set of all the action probabilities in time instant t in the tree as $\mathcal{P}(t)$. Then, by going through all the four paths that lead to four individual actions at level 2, and by following the action probability updating rules of the HDPA, we can calculate the expected probability of choosing the optimal action at level 2 as:

$$\begin{aligned}
&E\left[p_{21}(t+1)|\mathcal{P}(t)\right]\\
=&p_{11}p_{21}(d_{21}(q_{11}(p_{21}+\Delta)+(1-q_{11})(p_{21}-\Delta))\\
&+(1-d_{21})p_{21})\\
&+p_{11}p_{22}(d_{22}(q_{11}(p_{21}+\Delta)+(1-q_{11})(p_{21}-\Delta))\\
&+(1-d_{22})p_{21})\\
&+p_{12}p_{23}p_{21}+p_{12}p_{24}p_{21}\\
=&p_{11}p_{21}(d_{21}q_{11}p_{21}-d_{21}q_{11}\Delta+d_{21}p_{21}-d_{21}\Delta\\
&-d_{21}q_{11}p_{21}+d_{21}q_{11}\Delta+p_{21}-d_{21}p_{21})\\
&+p_{11}p_{22}(d_{22}q_{11}p_{21}-d_{22}q_{11}\Delta+d_{22}p_{21}-d_{22}\Delta\\
&-d_{22}q_{11}p_{21}+d_{22}q_{11}\Delta+p_{21}-d_{22}p_{21})\\
&+p_{12}p_{21}\\
=&p_{11}p_{21}d_{21}\Delta(2q_{11}-1)+p_{11}p_{21}p_{21}\\
&+p_{11}p_{22}d_{22}\Delta(2q_{11}-1)+p_{11}p_{22}p_{21}\\
&+p_{12}p_{21}\\
=&p_{11}\Delta(p_{21}d_{21}+p_{22}d_{22})(2q_{11}-1)+p_{21}.
\end{aligned}$$

Note that in the above expression, we have omitted all the time instant information to keep the expression simpler and neat. The complete version should be:

$$\begin{aligned}
&E\left[p_{21}(t+1)|\mathcal{P}(t)\right]\\
=&p_{11}(t)\Delta\Big(p_{21}(t)d_{21}+p_{22}(t)d_{22}\Big)\Big(2q_{11}(t)-1\Big)+p_{21}(t).
\end{aligned}$$

Following the same manner in which we calculated the above $E\left[p_{21}(t+1)|\mathcal{P}(t)\right]$, we are able to get the generalized formula for the expected probability of choosing the optimal action at level k:

$$E\left[p_{\{k,j_k^*\}}(t+1)|\mathcal{P}(t)\right] = p_{\{k,j_k^*\}}(t) + \left(\prod_{l=1,...,k-1} p_{\{l,j_l^*\}}(t)\right)$$

$$\sum_{j=1,2} p_{\{k,j\}}(t) d_{\{k,j\}} \Delta(2q_{\{k-1,j^*_{k-1}\}}(t)-1)\Bigg),$$

which can be proven by an inductive argument. The difference between $E\left[p_{\{k,j^*_k\}}(t+1)\right]$ and $p_{\{k,j^*_k\}}(t)$, is thus:

$$\begin{aligned} &Diff_{p_{\{k,j^*_k\}}(t)} = E[p_{\{k,j^*_k\}}(t+1)|P(t)] - p_{\{k,j^*_k\}}(t) \\ =&\Bigg(\prod_{l=1,...,k-1} p_{\{l,j^*_l\}}(t)\Bigg) \\ &\Bigg(\sum_{j=1,2} p_{\{k,j\}}(t) d_{\{k,j\}} \Delta(2q_{\{k-1,j^*_{k-1}\}}(t)-1)\Bigg). \end{aligned}$$

As $p_{\{k,j\}}(t) > 0$ and $\hat{d}_{\{k,j\}}(t) > 0$ for all t, k, and j, we clearly see that if $\forall t > t_0$, $q_{\{k,j^*_k\}}(t) > \frac{1}{2}$, then $Diff_{p_{\{k,j^*_k\}}(t)} > 0$, and the sequence $p_{\{k,j^*_k\}}$ with $t > t_0$ is a sub-martingale. Therefore, we only need to let $1-\delta = \frac{1}{2}$, then, by Theorem 2, there exists a time instant t_0, such that for all $t > t_0$ we have:

$$q_{\{k,j^*_k\}}(t) > \frac{1}{2}.$$

Theorem 3 is thus proven. □

D. The ϵ-Optimality of the HDPA

Finally, we show the ϵ-Optimality of the HDPA.

Theorem 4. *For all stationary stochastic Environments, the HDPA is ϵ-optimal, i.e., the HDPA will converge to the optimal path $\mathcal{P}^* = \{p_{\{k,j^*_k\}}\}, k = \{1,...,K\}$. Formally, given any $1-\delta > \frac{1}{2}$, there exists a $\Delta \in (0,1)$ and a time instant $t_0 < \infty$, such that for all $\Delta < \Delta_0$ and for all time instants $t > t_0$, we have $q_{\{k,j^*_k\}}(t) > 1-\delta, \forall k$, and the quantity:*

$$Pr\left\{p_{\{k,j^*_k\}}(\infty) = 1\right\} \to 1, \text{ where } k \in \{1,...,K-1\}.$$

Proof. We can interpret Theorem 4 as follows: If the probability of choosing the optimal action in each level converges to unity, then the entire tree will converge to the optimal path, which consists of the optimal nodes from each level. We thus can follow the level-wise approach to prove Theorem 4.

We again refer to the simplified tree and notations in Fig. 1 and consider the first two levels in the LA hierarchy, i.e., when $k = 1$. We are to prove that $Pr\{p_{11}(\infty) = 1\} \to 1$, and the proof is essentially the same as how we proved that a flat DPA converges to the optimal action in [36]. Consequently, the proof can be based on the sub-martingale convergence theory, and we can utilize a Regular function to indirectly study the convergence probability. Firstly, as per Theorem 3, $p_{11}(\infty)$ is a sub-martingale. According to the sub-martingale convergence theory, $p_{11}(\infty) = 0$ or 1. Secondly, the convergence probability $Pr\{p_{11}(\infty) = 1\}$ can be written as $Pr\{P_{01}(\infty) = e_m\}$, where e_m is the unit vector, and m is the index of element which corresponds to the optimal action. At node $\mathcal{A}_{\{0,1\}}$, $e_m = e_1 = [1,0]$, as the optimal action is the first action. Therefore, proving $Pr\{p_{11}(\infty) = 1\} \to 1$ is equivalent to proving $\Gamma(P_{01}) = Pr\{P_{01}(\infty) = e_1\} \to 1$.

To prove this, we shall use the theory of Regular functions, and follow an argument similar to the one given in [36]. Let $\Phi(P)$ be a function of P, and we define an operator U as:

$$U\Phi(P) = E[\Phi(P(t+1))|P(t) = P].$$

Then, we repeatedly apply U, resulting in the expression:

$$U^t\Phi(P) = E[\Phi(P(t))|P(0) = P].$$

If $\Phi(P) = U\Phi(P) = U^2\Phi(P) = ... = U^\infty\Phi(P)$, we call $\Phi(P)$ a regular function of P. If $\Phi(P) \geq U\Phi(P) \geq U^2\Phi(P) \geq ... \geq U^\infty\Phi(P)$, $\Phi(P)$ is a super-regular function of P, and when $\Phi(P) \leq U\Phi(P) \leq U^2\Phi(P) \leq ... \leq U^\infty\Phi(P)$, $\Phi(P)$ is a sub-regular function of P. Furthermore, if $\Phi(P)$ satisfies the boundary conditions:

$$\Phi(e_m) = 1 \text{ and } \Phi(e_j) = 0, (\text{for } j \neq m), \tag{1}$$

then, as per the definition of Regular functions and the sub-martingale convergence theory, we have:

$$\begin{aligned} U^\infty\Phi(P) &= E[\Phi(P(\infty))|P(0) = P] \\ &= \sum_{j=1,2} \Phi(e_m) Pr\{P(\infty) = e_j|P(0) = P\} \\ &= Pr\{P(\infty) = e_m|P(0) = P\} \\ &= \Gamma(P). \end{aligned} \tag{2}$$

Eq. (2) shows that $\Gamma(P)$ is exactly the function $\Phi(P)$ upon which U is applied an infinite number of times, and this function can be lower/upper bounded by $\Phi(P)$ if $\Phi(P)$ is a sub-regular/super-regular.

Our goal is to find a proper sub-regular function to serve as the lower bound $\Gamma(P_{01})$. We solve this by first finding a corresponding super-regular function of P_{01}. Let us consider a specific instantiation of Φ to be the function Φ_1 and:

$$\Phi_1(P_{01}) = e^{-x_1 p_{11}},$$

where x_1 is a positive constant. It follows that, under the HDPA, we have:

$$\begin{aligned} &U(\Phi_1(P_{01})) - \Phi_1(P_{01}) \\ &= E[\Phi_1(P_{01}(t+1))|\mathcal{P}(t)] - \Phi_1(P_{01}(t)) \\ &= E\left[e^{-x_1 p_{11}(t+1)}|P_{01}(t)\right] - e^{-x_1 p_{11}(t)} \\ &= \sum_{j=1,2} e^{-x_1(p_{11}+\Delta)} p_{1j} d_{1j} q_{01} \\ &\quad + \sum_{j=1,2} e^{-x_1(p_{11}-\Delta)} p_{1j} d_{1j} (1-q_{01}) \\ &\quad + \sum_{j=1,2} e^{-x_1 p_{11}} p_{1j}(1-d_{1j}) - e^{-x_1 p_{11}} \\ &= \sum_{j=1,2} p_{1j} d_{1j} e^{-x_1 p_{11}} \Bigg(q_{01}\Big(\left(e^{-x_1\Delta} - e^{x_1\Delta}\right) \\ &\qquad + (e^{x_1\Delta} - 1)\Big)\Bigg). \end{aligned} \tag{3}$$

In the above equation, time instant (t) has been omitted from $p_{1j}(t)$, $p_{11}(t)$, and $q_{01}(t)$. We now need to determine a proper value for x_1 such that $\Phi_1(P)$ is super-regular, i.e.,

$$U(\Phi_1(P_{01})) - \Phi_1(P_{01}) \leq 0,$$

which is equivalent to solving the following inequality:

$$q_{01}\left(e^{-x_1\Delta}-e^{x_1\Delta}\right)+\left(e^{x_1\Delta}-1\right)\leq 0.$$

By Taylor expansion, the above equation can be approximated:

$$\begin{aligned}
&q_{01}\left(1-x_1\Delta+\frac{x_1^2\Delta^2}{2}-1-x_1\Delta-\frac{x_1^2\Delta^2}{2}\right)\\
&+1+x_1\Delta+\frac{x_1^2\Delta^2}{2}-1\leq 0\\
\Rightarrow& x_1(\frac{x_1\Delta}{2}+1-2q_{01})\leq 0\\
\Rightarrow& 0\leq x_1\leq\frac{2(2q_{01}-1)}{\Delta}. \qquad (4)
\end{aligned}$$

Let us now introduce a new function, $\phi_1(P_{01})$, which satisfies the boundary conditions as follows:

$$\phi_1(P_{01})=\frac{1-e^{-x_1p_{11}}}{1-e^{-x_1}}=\begin{cases}1, \text{ when } P_{01}=e_1,\\ 0, \text{ when } P_{01}=e_2.\end{cases}$$

where x_1 is similar to how it is defined in $\Phi_1(P_{01})$. It follows that, if $\Phi_1(P_{01})=e^{-x_1p_{11}}$ is super-regular (sub-regular), then $\phi_1(P_{01})=\frac{1-e^{-x_1p_{11}}}{1-e^{-x_1}}$ is a sub-regular (super-regular) [2]. The definition of x_1 that renders $\Phi_1(P_{01})$ to be super-regular, makes $\phi_1(P_{01})$ to be sub-regular. Following the property of Regular functions, it follows that:

$$\Gamma(P_{01})\geq\phi_1(P_{01})=\frac{1-e^{-x_1p_{11}}}{1-e^{-x_1}}. \qquad (5)$$

As Eq. (5) holds for every x_1 bounded by Eq. (4), we can choose the largest value $x_{1max}=\frac{2(2q_{01}-1)}{\Delta}$. When $\Delta\to 0$, we have $x_{1max}\to\infty$, which renders $\phi_1(P_{01})\to 1$, hence $\Gamma(P_{01})\to 1$. Thus, $Pr\left(p_{11}(\infty)=1\right)\to 1$ under the HDPA.

The above proof methodology can be applied to higher levels of the HDPA hierarchy. Take the simplest hierarchical DPA with $K=2$ for an example, again, we refer to Fig. 1 for the simplified notations, and we are to prove that $\Gamma(P_{11})=Pr\left\{P_{11}(\infty)=e_1\right\}=Pr\left\{p_{21}(\infty)=1\right\}\to 1$.

Let us consider another specific instantiation of Φ to be:

$$\Phi_2(P_{11})=e^{-x_2p_{21}},$$

where x_2 is a positive constant. Under the HDPA, we have:

$$\begin{aligned}
&U(\Phi_2(P_{11}))-\Phi_2(P_{11})\\
=&E[\Phi_2(P_{11}(t+1))|\mathcal{P}(t)]-\Phi_2(P_{11}(t))\\
=&E\left[e^{-x_2p_{21}(t+1)}|\mathcal{P}(t)\right]-e^{-x_2p_{21}(t)}\\
=&e^{-x_2(p_{21}+\Delta)}(p_{11}p_{21}d_{21}q_{11})+e^{-x_2(p_{21}-\Delta)}(p_{11}p_{21}d_{21}(1-q_{11}))\\
&+e^{-x_2p_{21}}(p_{11}p_{21}(1-d_{21}))\\
&+e^{-x_2(p_{21}+\Delta)}(p_{11}p_{22}d_{22}q_{11})+e^{-x_2(p_{21}-\Delta)}(p_{11}p_{22}d_{22}(1-q_{11}))\\
&+e^{-x_2p_{21}}(p_{11}p_{22}(1-d_{22}))\\
&+e^{-x_2p_{21}}(p_{12}p_{23})+e^{-x_2p_{21}}(p_{12}p_{24})\\
&-e^{-x_2p_{21}}\\
=&e^{-x_2p_{21}}\Bigg[p_{11}p_{21}\left(e^{-x_2\Delta}d_{21}q_{11}+e^{x_2\Delta}d_{21}(1-q_{11})+(1-d_{21})\right)\\
&+p_{11}p_{22}\left(e^{-x_2\Delta}d_{22}q_{11}+e^{x_2\Delta}d_{22}(1-q_{11})+(1-d_{22})\right)+p_{12}\Bigg]\\
&-e^{-x_2p_{21}}\\
=&e^{-x_2p_{21}}\Bigg[p_{11}\sum_{j=1,2}p_{2j}\left(e^{-x_2\Delta}d_{2j}q_{11}\right.\\
&\left.+e^{x_2\Delta}d_{2j}(1-q_{11})+(1-d_{2j})\right)+p_{12}\Bigg]-e^{-x_2p_{21}}\\
=&e^{-x_2p_{21}}\Bigg[p_{11}\sum_{j=1,2}p_{2j}\left(e^{-x_2\Delta}d_{2j}q_{11}\right.\\
&\left.+e^{x_2\Delta}d_{2j}(1-q_{11})-d_{2j}\right)+p_{11}\sum_{j=1,2}p_{2j}+p_{12}\Bigg]-e^{-x_2p_{21}}\\
=&e^{-x_2p_{21}}\Bigg[p_{11}\sum_{j=1,2}p_{2j}\left(e^{-x_2\Delta}d_{2j}q_{11}\right.\\
&\left.+e^{x_2\Delta}d_{2j}(1-q_{11})-d_{2j}\right)+1\Bigg]-e^{-x_2p_{21}}\\
=&e^{-x_2p_{21}}\left[p_{11}\sum_{j=1,2}p_{2j}d_{2j}\left(e^{-x_2\Delta}q_{11}+e^{x_2\Delta}(1-q_{11})-1\right)\right)\Bigg]\\
=&e^{-x_2p_{21}}\Bigg[p_{11}\sum_{j=1,2}p_{2j}d_{2j}\left(q_{11}\left(e^{-x_2\Delta}-e^{x_2\Delta}\right)+\right.\\
&\left.(e^{x_2\Delta}-1)\right)\Bigg] \qquad (6)
\end{aligned}$$

Just as in Eq. (3), in the above Eq. (6), the time instant (t) has been omitted from $p_{2j}(t)$, $p_{11}(t)$, and $q_{11}(t)$, to make the notation less cumbersome. In order for $\Phi_2(P_{11})$ to be super-regular, we need:

$$\begin{aligned}
&U(\Phi_2(P_{11}))-\Phi_2(P_{11})\leq 0\\
\Rightarrow& q_{11}\left(e^{-x_2\Delta}-e^{x_2\Delta}\right)+\left(e^{x_2\Delta}-1\right)\leq 0\\
\Rightarrow& 0\leq x_2\leq\frac{2(2q_{11}-1)}{\Delta}. \qquad (7)
\end{aligned}$$

The same x_2 will render $\phi_2(P_{11})=\frac{1-e^{-x_2p_{21}}}{1-e^{-x_2}}$ to be sub-regular. Clearly, $\phi_2(P_{11})$ meets the boundary condition, thus:

$$\Gamma(P_{11})\geq\phi_2(P_{11})=\frac{1-e^{-x_2p_{21}}}{1-e^{-x_2}}. \qquad (8)$$

Similarly, when $\Delta\to 0$, we have $x_{2max}=\frac{2(q_{11}-1)}{\Delta}\to\infty$, which renders $\phi_2(P_{11})\to 1$, whence $\Gamma(P_{11})=Pr\{p_{21}(\infty)=1\}\to 1$ under the HDPA.

As explained above, the overall proof of the entire tree follows by a simple inductive argument. Defining x_k as a positive constant, we can generalize Eq. (6) to any level k:

$$\begin{aligned}
&U(\Phi_k(P_{\{k-1,j^*_{k-1}\}}))-\Phi_k(P_{\{k-1,j^*_{k-1}\}})\\
=&E[\Phi_k(P_{\{k-1,j^*_{k-1}\}}(t+1))|\mathcal{P}(t)]-\Phi_k(P_{\{k-1,j^*_{k-1}\}}(t))\\
=&E\left[e^{-x_kp_{\{k,j^*_k\}}(t+1)}|\mathcal{P}(t)\right]-e^{-x_kp_{\{k,j^*_k\}}(t)}\\
=&e^{-x_kp_{\{k,j^*_k\}}}\left(\prod_{l=0,\ldots,k-1}p_{\{l,j^*_l\}}(t)\right)\left(\sum_{j=1,2}p_{\{k,j_k\}}d_{\{k,j\}}\right.\\
&\left.q_{\{k-1,j^*_{k-1}\}}\left(e^{-x_k\Delta}-e^{x_k\Delta}\right)+(e^{x_k\Delta}-1)\right)\Bigg), \qquad (9)
\end{aligned}$$

and the conclusion is the same: when $\Delta\to 0$, we have $x_{kmax}=\frac{2\left(2q_{\{k-1,j^*_{k-1}\}}-1\right)}{\Delta}\to\infty$, which renders $\phi_k\left(P_{\{k-1,j^*_{k-1}\}}\right)\to 1$. Hence $\Gamma(P_{\{k-1,j^*_{k-1}\}})=Pr\{p_{\{k,j^*_k\}}(\infty)=1\}\to 1$ under the HDPA.

As the LA converges to the optimal action at each level, the HDPA converges the optimal path, proving Theorem 4. □

V. Numerical Results

To demonstrate the performance of the proposed HDPA scheme, we carried out extensive simulations for different Environments with "many" actions. To ensure the credibility of our simulations, we increased the number of experiments and the convergence criteria compared to the experiments in [31]. As highlighted earlier, one of the drawbacks of the HCPA is that it has a sluggish increase in updating the action probabilities when the probability to be increased approaches unity. As opposed to this, we expected that the HDPA, which has a constant increment in the action probability, would have significantly faster convergence than that of the HCPA when the convergence criterion is close to unity. Our results presented below, demonstrated that the HDPA, indeed, required significantly fewer iterations for cases requiring high accuracy.

A. The Simulation Environments

We conducted simulations for Environments with 16, 32, 64, 128, 256 and 512 actions. We configured the simulation Environments for the 16, 32, and 64 actions on the benchmark action reward probabilities established in [31], which are listed in Tab. I. The table lists 64 reward probabilities. The parameter j_K in Tab. I indicates the index of the action at depth K in the HDPA structure, and $\Omega = Pr\{\beta = 0\}$ shows the probability of that action obtaining a Reward from the Environment. We utilized the first 16 probabilities as the 16-action Environment, i.e., $j_K \in \{1, ..., 16\}$. Likewise, for the 32-action Environment, we used $j_K \in \{1, ..., 32\}$, and so on. For the 128 and 256 actions Environments, we uniformly generated 128 and 256 reward probabilities between zero and unity, visualized in Fig. 2 and Fig. 3, respectively. Note that for 128, 256 and 512 actions, the reward probabilities utilized were distinct from those used for the Environments in [31].

B. The Algorithms' Learning Parameters

From the mathematical proof above and the established theory of VSSA, we know that when the learning parameter, Δ, is sufficiently small, the HDPA will converge to the action that has the maximum reward probability with a probability close to unity. The same is true when it concerns the value of λ for the HCPA. The respective values for λ and Δ determine how quickly the LA achieves convergence. The higher the learning parameters are configured, the faster the algorithms converge. However, if the learning parameters are configured too high, the algorithm might not converge to the optimal action with high probability. Therefore, the tuning of these parameters is a trade-off between the accuracy of finding the optimal action and the speed of convergence.

To find the best values for λ and Δ, we utilized a top-down approach[10]. More specifically, we decreased the value of the learning parameters in a step-wise manner with two decimals precision until their configured values made the LA achieve 100% accuracy in converging to the optimal action for all the number of experiments arranged. Consequently, the values of the learning parameters that fulfilled these criteria represented the assumed "best" values for λ and Δ given the distinct Environments used in the simulations. We emphasize that tuning the values of λ and Δ are challenging because the Environments's stochastic nature can cause uncertainty in the values of λ and Δ. Moreover, although we obtained the values for λ and Δ through extensive testing, it is not true that the HDPA will "always" select the optimal action because the convergence is, indeed, in probability due to the ϵ-optimal property. The reader should also note that the λ and Δ values are dependent on the Environment's reward probabilities and that the best values of λ and Δ vary from Environment to Environment. Therefore, we refer to the values determined for λ and Δ in this paper as the "best" learning parameters for the given Environments, and not the "optimal" ones.

[10]This issue is application dependent, and there is no hard-and-fast rule to determine this. We thank the anonymous Referee who pointed this out to us.

C. Average Number of Iterations

The average number of iterations required before *convergence* (to a convergence threshold, T), is an established parameter for evaluating a learning scheme's. efficiency.

We first address the simulation results presented in Table II. For these simulations, we conducted 600 experiments and configured the convergence criterion to be 0.992. Thus, we affirmed that the LA had converged when it achieved a 0.992 probability of choosing one of the actions in its action probability vector. All the results presented in Table II were based on *all 600* experiments converging to the optimal action.

For the Environment with 16 actions, the "best" learning parameters were $\lambda = 0.0043$ and $\Delta = 0.0011$. For the Environment with 32 and 64 actions, the best learning parameters were $\lambda = 0.00057$ $\lambda = 3.6e{-}5$, $\Delta = 0.00015$ and $\Delta = 9.9e{-}6$, respectively. The "best" obtained values for the Environment with 128 actions were $\lambda = 3.9e{-}5$ and $\Delta = 9.7e{-}6$. For 256 actions, we obtained $\lambda = 5.9e{-}6$ and $\Delta = 1.5e{-}6$ as the "best" learning parameters. For the 512-action case, visualized in Fig. 8, we used $\lambda = 7.9e{-}5$ and $\Delta = 1.7e{-}5$. From the these, we can observe that the optimal action is further away from the sub-optimal one. Therefore, the Environment is "simpler" than, e.g., the 64, 128 and 256-action cases, because the algorithms required less number of iterations for this Environment. Thus, the hardness of the Environment can impact the number of iterations more than the number of actions.

Table II includes both algorithms' results, and lists both the Mean and the Standard Deviation (Std) of the number of iterations required before convergence. Let us first consider the cases with 16, 32, and 64 actions based on the benchmark probabilities in Table I. The HDPA required just 64%, 60%, and 59% of the total number of iterations that the HCPA required for the 16, 32, and 64 actions, respectively. The benefit of HDPA over HCPA increased with the number of actions. Observing the Std, the HDPA had more variation in the number of iterations for all three environments.

Considering the results for 128 actions (with action reward probabilities depicted in Fig. 2), the HDPA converged within

TABLE I
BENCHMARK ACTION REWARD PROBABILITIES ESTABLISHED IN [31].

j_K	Ω	j_K	Ω	j_K	Ω	j_K	Ω	j_K	Ω	j_K	Ω	j_K	Ω	j_K	Ω
1	0.3934	9	0.0333	17	0.7362	25	0.02152	33	0.6214	41	0.0970	49	0.2413	57	0.4763
2	0.9902	10	0.4323	18	0.7603	26	0.2399	34	0.9777	42	0.1319	50	0.1714	58	0.4446
3	0.4883	11	0.6926	19	0.5142	27	0.7509	35	0.4232	43	0.1738	51	0.8512	59	0.9617
4	0.5768	12	0.3474	20	0.2273	28	0.8773	36	0.02773	44	0.8901	52	0.9791	60	0.0329
5	0.2023	13	0.6152	21	0.6080	29	0.4962	37	0.1255	45	0.3511	53	0.7443	61	0.5004
6	0.2390	14	0.0900	22	0.4791	30	0.5649	38	0.5650	46	0.8945	54	0.3469	62	0.3784
7	0.5887	15	0.0850	23	0.9339	31	0.9202	39	0.1660	47	0.6133	55	0.8707	63	0.6553
8	0.8894	16	0.5652	24	0.3808	32	0.1335	40	0.0148	48	0.4813	56	0.3863	64	0.9737

Fig. 2. The action reward probabilities for the 128 actions Environment.

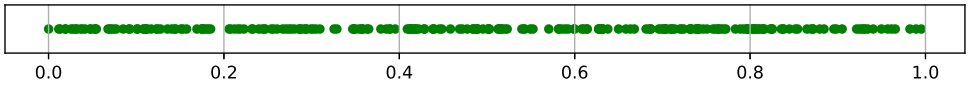


Fig. 3. The action reward probabilities for the 256 actions Environment.

TABLE II
THE SIMULATION RESULTS OBTAINED FOR THE VARIOUS ENVIRONMENTS.

Nr. of Actions	HCPA		HDPA	
16	Mean:	1,366.61	Mean:	868.25
	Std:	121.14	Std:	135.50
32	Mean:	10,281.84	Mean:	6,172.38
	Std	681.82	Std:	744.84
64	Mean:	169,839.67	Mean:	100,638.41
	Std:	13,687.48	Std:	17,653.41
128	Mean:	155,088.62	Mean:	97,795.59
	Std:	10,613.21	Std:	13,266.12
256	Mean:	1,039,215.58	Mean:	632,985.27
	Std:	88,744.96	Std:	70,978.51
512	Mean:	95,354.61	Mean:	78,779.84
	Std:	13,099.73	Std:	15,418.82

approximately 97,800 iterations on average, while the HCPA required 155,000 iterations. Comparing the algorithms' Std, we again observe that the HDPA had more variation compared with that of the HCPA. This indicates that HDPA is slightly more unpredictable in its convergence iterations. As we can observe for the 256 case, the HDPA required significantly fewer iterations than the HCPA. More specifically, the HDPA achieved convergence with just 60% of the iterations that the HCPA required, which clearly demonstrated the advantage of the HDPA over HCPA in the above-studied configurations.

D. Illustrative Details of Convergence Analysis

In this subsection, we present more convergence details for the Environments with 64 actions. For these simulations, we conducted 1000 experiments with 0.999 as the convergence criterion. The "best" learning parameters for these simulations, found with the top-down approach, were $\lambda = 3.8e{-}5$ and $\Delta = 9.4e{-}6$, respectively. In Fig. 4, a cumulative representation of the number of iterations required with the different schemes are presented. As we can observe, the HDPA required significantly fewer iterations, and we see that approximately 90% of the experiments converged within approximately 140,000 iterations. In comparison, none of the HCPA experiments required less than 200,000 iterations.

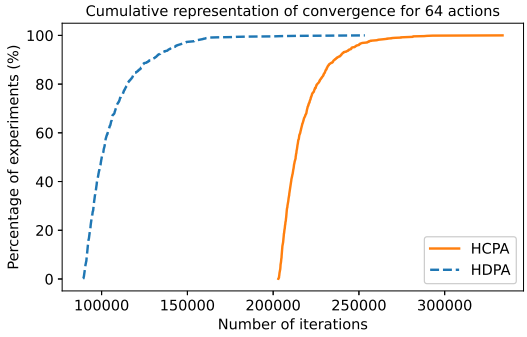


Fig. 4. Cumulative representation of the percentage of experiments converging within a certain number of iterations. The figure is based on 1,000 experiments, with 0.999 as the convergence criterion for 64 actions.

In Fig. 5, we present the number of iterations required for the HCPA and the HDPA through a scatter representation. We can observe that some of the HDPA experiments required more iterations than the HCPA, and that some of them coincide with the number of iterations as the HCPA. However, these are outliers, and most of the experiments are concentrated at around 100,000 for the HDPA. For the HCPA we see that the

number of iterations is located at around 200,000. The reason that the iteration numbers in most of the experiment instances are close to the actual minimum number is that the numbers of iterations become close to each other after the action selection probabilities reach a certain level. In these cases, it is very likely that the LA choose the optimal actions, and that they also obtain a reward for choosing these actions. Due to the pursuit concept, the system rewarded the current best actions even if a sub-optimal action was chosen and rewarded.

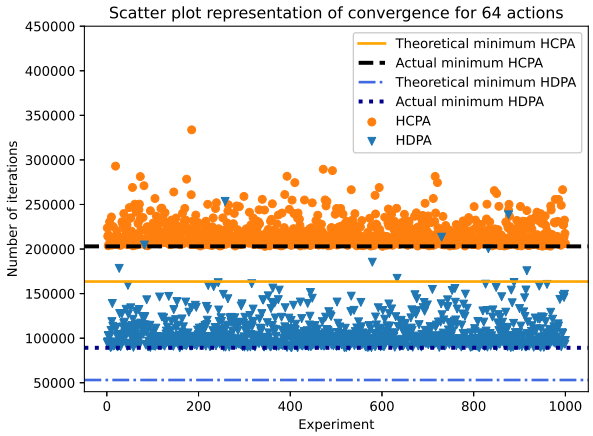


Fig. 5. Scatter plot of the experiment number and the number of iterations it took before the LA converged for that experiment. The figure is based on 1,000 experiments, with 0.999 as the convergence criterion for 64 actions.

E. Performance for Different Convergence Criteria

In Fig. 6 and Fig. 7, we present the performance of the HCPA and the HDPA for different convergence criteria for 64 actions. In these simulations, we conducted 100 experiments, where we used the top-down approach for finding the "best" λ and Δ for each individual convergence criterion. The "best" λ and Δ for the convergence criteria between 0.90 and 0.99 can be made available to the interested readers. For the convergence criteria between 0.990 and 0.999, the "best" learning parameters were found to be $\lambda = 5.3e{-}5$ and $\Delta = 1.0e{-}5$ for all convergence criteria[11]. The reason why these learning parameters remained constant, is that when the LA reached such a high probability level as 0.990, it rarely changed to another action than the one it was currently pursuing. Consequently, the values for λ and Δ that we found, represented the highest (fastest) learning parameters for all convergence criteria above 0.990.

In Fig. 6, we can observe that the HCPA required less iterations on average for the convergence criteria between 0.9 and 0.97. However, for the convergence criteria of 0.98 and 0.99, the HDPA had fewer required iterations on average. Consequently, we observed that the HDPA was more efficient as the convergence requirement was configured closer to unity.

[11]The reader should note that the "best" values for λ and Δ found here differ from the "best" ones reported in Section V-C and V-D because the numbers of experiments are different. Indeed, the 1,000 experiments stipulated in Section V-D set a higher requirement to the learning parameters than the 100 experiments in Section V-E, and thus required more conservative learning.

The effectiveness of the HDPA for higher convergence criteria was further verified in Fig. 7, where the HDPA had fewer required iterations on average for all convergence requirements. Additionally, it is clear from Fig. 7 that the difference between the two schemes increased as the criterion increased.

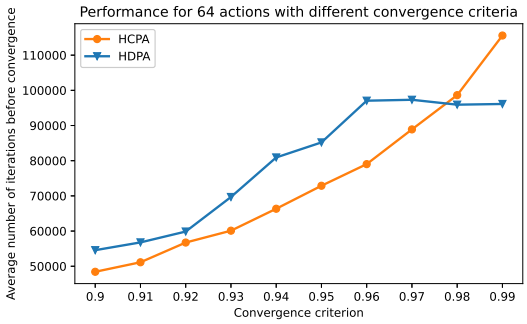


Fig. 6. The average number of iterations required before convergence for 100 experiments with different convergence criteria from 0.90 to 0.99 for a 64 actions Environment.

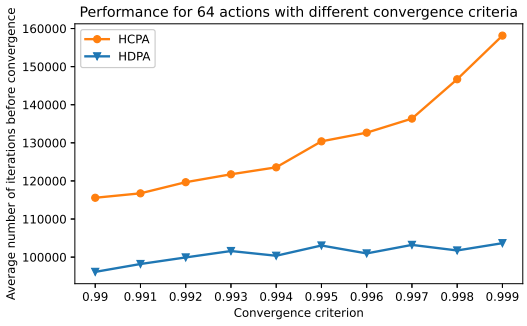


Fig. 7. The average number of iterations required before convergence for 100 experiments with different convergence criteria from 0.990 to 0.999 for a 64 actions Environment.

F. Comparison with Other Competing LA

Before we conclude this paper, it is pertinent that we compare our results with another set of algorithms[12]. These papers [37]–[39], (referred to as AlgJu1, AlgJu2, and AlgJu3) have all been written by the same team of researchers, and it is prudent to compare them quantitatively and qualitatively.

Our first remark is that these papers are brilliant within the field of LA. We have implemented all of them and found that they are very competitive even against the best-reported schemes. Of particular interest, however, is the scheme AlgJu3 presented in [39], which is noticeably superior to AlgJu1 and AlgJu2 from [37] and [38], respectively. In the interest of brevity and space, we merely report below the results comparing our work with the paper of [39], while the comparisons with [37] and [38] are included in the PhD thesis [40].

[12]We were unaware of these papers, when we submitted the initial version of the paper. We are extremely grateful to the anonymous Referee who directed us to them. This section is dedicated to such a mutual comparison.

Fig. 8. The action reward probabilities for the 512 actions Environment.

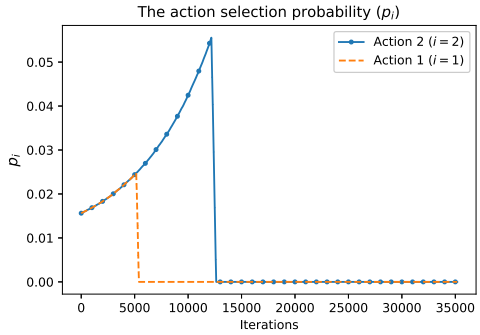


Fig. 9. The performance for a 64-action Environment where the optimal action changes over time. After about 13,000 iterations both α_1 and α_2 are discarded from the competition because of the Environment's non-stationarity.

Nr. of Actions	HDPA		AlgJu3	
10 (E3 in [39])	Mean:	6,813.089	Mean:	3,721.99 (6,134.39)
	Std:	1274.38	Std:	235.14 (159.38)
	Acc:	100%	Acc:	99.2% (100%)
	Δ:	0.00026	n:	59 (100)
64 (Env. in [31])	Mean:	104,884.60	Mean:	35,713.68
	Std:	18393.40	Std:	532.42
	Acc:	100%	Acc:	100%
	Δ:	$9.5e^-6$	n:	100
512 (in Fig. 8)	Mean:	79,623.64	Mean:	46,004.62
	Std:	16,432.28	Std:	587.53
	Acc:	100%	Acc:	100%
	Δ:	$1.7e^-5$	n:	10

TABLE III
A COMPARISON BETWEEN THE HDPA AND ALGJU3 BASED ON 1,000 EXPERIMENTS.

The methodology that is used in AlgJu3 has the potential of making it the most superior algorithm. However, the underlying principle is a two-edged sword as explained below.

- The principle which renders this paper competitive is that after a sufficiently large number of iterations it is able to distinguish between "the boys and the men". In this way, the algorithm decides to discard many of the rather inferior actions, resulting in a brilliant paradigm of learning within a *smaller* action space. This is the reason why the method becomes so competitive.
- The negative side of such a decision is that if such an action-pruning task is undertaken after a large number of iterations, it is, indeed, not profitable. But if it is undertaken before this "large" number of iterations, in many cases the scheme, unfortunately, discards some of the more superior actions. While this is mostly an infrequent occurrence, in reality, when the actions are highly competitive, it does occur to a noticeable extent by which some of the best actions are discarded.
- The above negative phenomenon, while being less likely in stationary Environments, is an almost-predominant occurrence in "non-stationary" Environments.
- While the above phenomenon can be mitigated by using a small enough learning parameter, the question of determining the size of the parameter is still open, other than by invoking a meta-learning scheme.

The results of comparing our algorithm with AlgJu3 [39] are given in Table III, where as mentioned above, the weaker schemes of AlgJu1 and AlgJu2 are omitted. The reader will observe that AlgJu3 is superior in most cases, but we have to emphasize that the results reported in the paper are not as stringent as those reported here because we have required a categoric 100% accuracy. AlgJu3 is arguably superior, but it has an impediment in that in some cases it discards the best action from the entire competition. Thus, *while it appears to be superior*, as explained in the above items, the truth of the matter is that its superiority is a result of operating within a diminished action space.

This comment has a major significance as mentioned in the third item above. The graph in Fig. 9, displays the action selection probability when α_1 is the best action and after about 13,000 iterations it discards both α_1 and α_2 because of the non-stationarity of the Environment.

VI. CONCLUSION

In this paper, we have thoroughly surveyed the strategies to enhance the speed and accuracy in the field of LA over the last six decades. By incorporating the major phenomena into a single machine, namely the HDPA, proposed in this paper, we are able to beat the state-of-the-art HCPA in terms of efficiency when the accuracy requirement is uniformly superior, e.g., above 0.99. In specifically, the HDPA combines VSSA probability updating functionality, discretizing the probability space and the Estimator phenomenon together into a hierarchical tree structuring with pursuit capabilities. We have explained the fine details of the algorithm and proven its ϵ-optimality through a formal, rigorous mathematical analysis. Our simulation results have demonstrated the advantage of the HDPA compared with the HCPA when the convergence criterion is close to unity. Indeed, the HDPA is arguably the best LA reported in the literature, to-date.

Acknowledgements: This work is supported by the project "Spacetime Vision: Towards Unsupervised Learning in the 4D World" financed by the EEA and Norway Grants 2014-2021 under the grant number EEA-RO-NO-2018-04.

REFERENCES

[1] M. L. Tsetlin, "Finite Automata and Modeling the Simplest Forms of Behavior," *Uspekhi Matem Nauk*, vol. 8, no.4, pp. 1–26, 1963.

[2] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*, ser. Dover Books on Electrical Engineering Series, Dover Publications, Incorporated, 2012.

[3] S. Lakshmivarahan, *Learning Algorithms Theory and Applications*, ed. 1, Springer-Verlag New York, 1981.

[4] M. A. L. Thathachar and P. S. Sastry, "Estimator Algorithms for Learning Automata," *Proceedings of the Platinum Jubilee Conference on Systems and Signal Processing*, Department of Electrical Engineering, Indian Institute of Science, 1986.

[5] R. Thapa, L. Jiao, B. J. Oommen, and A. Yazidi, "A learning automaton-based scheme for scheduling domestic shiftable loads in smart grids," *IEEE Access*, vol. 6, pp. 5348–5361, 2018.

[6] A. Yazidi, I. Hassan, H. L. Hammer, and B. J. Oommen, "Achieving fair load balancing by invoking a learning automata-based two-time-scale separation paradigm," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3444–3457, 2021.

[7] S. Sahoo, B. Sahoo, and A. K. Turuk, "A Learning Automata-Based Scheduling for Deadline Sensitive Task in The Cloud," *IEEE Transactions on Services Computing*, vol. 14, no. 6, pp. 1662–1674, Nov. 2021.

[8] L. Zhu, K. Huang, Y. Hu, and X. Tai, "A Self-Adapting Task Scheduling Algorithm for Container Cloud Using Learning Automata," *IEEE Access*, vol. 9, pp. 81 236–81 252, 2021.

[9] R. R. Rout, G. Lingam, and D. V. L. N. Somayajulu, "Detection of malicious social bots using learning automata with url features in twitter network," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 4, pp. 1004–1018, 2020.

[10] H. Guo, S. Li, K. Qi, Y. Guo, and Z. Xu, "Learning automata based competition scheme to train deep neural networks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 2, pp. 151–158, 2020.

[11] Z. Zhang, D. Wang, and J. Gao, "Learning automata-based multiagent reinforcement learning for optimization of cooperative tasks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4639–4652, 2021.

[12] C. Di, B. Zhang, Q. Liang, S. Li, and Y. Guo, "Learning automata-based access class barring scheme for massive random access in machine-to-machine communications," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6007–6017, 2019.

[13] S. Tanwar, S. Tyagi, N. Kumar, and M. S. Obaidat, "La-mhr: Learning automata based multilevel heterogeneous routing for opportunistic shared spectrum access to enhance lifetime of wsn," *IEEE Systems Journal*, vol. 13, no. 1, pp. 313–323, 2019.

[14] B. El Khamlichi, D. H. N. Nguyen, J. El Abbadi, N. W. Rowe, and S. Kumar, "Learning automaton-based neighbor discovery for wireless networks using directional antennas," *IEEE Wireless Communications Letters*, vol. 8, no. 1, pp. 69–72, 2019.

[15] X. Deng, Y. Jiang, L. T. Yang, L. Yi, J. Chen, Y. Liu, and X. Li, "Learning-automata-based confident information coverage barriers for smart ocean internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9919–9929, 2020.

[16] Z. Yang, Y. Liu, Y. Chen, and L. Jiao, "Learning automata based q-learning for content placement in cooperative caching," *IEEE Transactions on Communications*, vol. 68, no. 6, pp. 3667–3680, 2020.

[17] R. O. Omslandseter, L. Jiao, Y. Liu, and B. John Oommen, "User grouping and power allocation in NOMA systems: A novel semi-supervised reinforcement learning-based solution," *Pattern Analysis and Applications*, July 2022.

[18] O.-C. Granmo, "The Tsetlin Machine – A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic," April 2018, arXiv:1804.01508.

[19] R. K. Yadav, L. Jiao, O.-C. Granmo, and M. Goodwin, "Robust interpretable text classification against spurious correlations using and-rules with negation," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, L. D. Raedt, Ed. International Joint Conferences on Artificial Intelligence Organization, July 2022, pp. 4439–4446.

[20] K. D. Abeyrathna, B. Bhattarai, M. Goodwin, S. Gorji, O.-C. Granmo, L. Jiao, R. Saha, and R. K. Yadav, "Massively Parallel and Asynchronous Tsetlin Machine Architecture Supporting Almost Constant-Time Scaling," In proceedings of *The Thirty-eighth International Conference on Machine Learning (ICML)*, 2021.

[21] L. Jiao, X. Zhang, O.-C. Granmo, and K. D. Abeyrathna, "On the Convergence of Tsetlin Machines for the XOR Operator," *IEEE Trans. Pattern Anal. Mach. Intell.*, accepted, Aug. 2022.

[22] X. Zhang, L. Jiao, O.-C. Granmo, and M. Goodwin, "On the Convergence of Tsetlin Machines for the IDENTITY- and NOT Operators," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6345–6359, 2022.

[23] S. Lakshmivarahan and M. A. L. Thathachar, "Absolutely Expedient Learning Algorithms for Stochastic Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 3, pp. 281–286, 1973.

[24] B. J. Oommen, "Absorbing and Ergodic Discretized Two-Action Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 2, pp. 282–293, 1986.

[25] B. J. Oommen and J. P. R. Christensen, "ϵ-Optimal Discretized Linear Reward-Penalty Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, no. 3, pp. 451–458, 1988.

[26] X. Zhang, B. J. Oommen and O.-C. Granmo, "The Design of Absorbing Bayesian Pursuit Algorithms and the Formal Analyses of Their ϵ-Optimality," *Pattern Analysis and Applications*, vol. 20, pp. 797–808, 2017.

[27] M. Agache and B. J. Oommen, "Generalized Pursuit Learning Schemes: New Families of Continuous and Discretized Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 6, pp. 738–749, 2002.

[28] J. K. Lanctot and B. J. Oommen, "Discretized Estimator Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1473–1483, 1992.

[29] B. J. 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.

[30] X. Zhang, O.-C. Granmo, B. J. Oommen, "Discretized Bayesian Pursuit - A New Scheme for Reinforcement Learning," *Advanced Research in Applied Artificial Intelligence*, vol. 7345, pp. 784–793, 2012.

[31] A. Yazidi, X. Zhang, L. Jiao, and B. J. 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*, vol. 31, no. 2, pp. 512–526, 2020.

[32] G. I. Papadimitriou, "Hierarchical Discretized Pursuit Nonlinear Learning Automata with Rapid Convergence and High Accuracy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 4, pp. 654–659, 1994.

[33] X. Zhang, B. J. Oommen, O.-C. Granmo, and L. Jiao, "Using the Theory of Regular Functions to Formally Prove the ϵ-Optimality of Discretized Pursuit Learning Algorithms," *Proceedings, Part I, of the 27th International Conference on Modern Advances in Applied Intelligence*, vol. 8481, pp. 379–388, 2014.

[34] K. Rajaraman and P. S. Sastry, "Finite Time Analysis of the Pursuit Algorithm for Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 4, pp. 590–598, 1996.

[35] X. Zhang, L. Jiao, B. J. 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*, vol. 31, no. 1, pp. 284–294, 2020.

[36] X. Zhang, B. J. Oommen, O.-C. Granmo, and L. Jiao, "A Formal Proof of the ϵ-Optimality of Discretized Pursuit Algorithms," *Applied Intelligence*, vol. 44, no. 2, pp. 282–294, 2016.

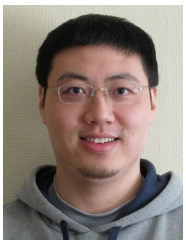
[37] J. Zhang, C. Wang, D. Zang, and M. Zhou, "Incorporation of Optimal Computing Budget Allocation for Ordinal Optimization Into Learning Automata," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 1008–1017, 2016.

[38] J. Zhang, C. Wang, and M. Zhou, "Fast and Epsilon-Optimal Discretized Pursuit Learning Automata," *IEEE Transactions on Cybernetics*, vol. 45, no. 10, pp. 2089–2099, 2015.

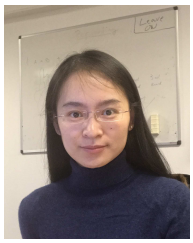
[39] J. Zhang, C. Wang, and M. Zhou, "Last-Position Elimination-Based Learning Automata," *IEEE Transactions on Cybernetics*, vol. 44, no. 12, pp. 2484–2492, 2014.

[40] R. O. Omslandseter, "On the Theory and Applications of Hierarchical Learning Automata and Object Migration Automata," *Ph. D. Thesis*, University of Agder, 2023.

Rebekka O. Omslandseter was born in Porsgrunn, Norway on January 15, 1995. She received her BE degree from University of Agder, Norway, in 2017. She is currently an integrated Ph. D. student at the University of Agder, and she is a scientific researcher at Centre of Artificial Intelligence Research (CAIR) in UiA. Her research interests include Learning Automata, Partitioning and Clustering Algorithms, Resource Allocation and Performance Evaluation for Communication and Energy Systems.



Lei Jiao (M'12-SM'18) received his BE degree from Hunan University, China, in 2005. He received his ME degree from Shandong University, China, in 2008. He obtained his PhD degree in Information and Communications Technology from University of Agder (UiA), Norway, in 2012. He is now an Associated Professor in the Department of Information and Communication Technology, UiA. His research interests include Reinforcement Learning, Learning Automata, Natural Language Processing, Resource Allocation for Communication and Energy Systems.



Xuan Zhang received her PhD degree in Information and Communication Technology from the University of Agder (UiA) in 2015. She has a Master's degree in Signal and Information Processing and a Bachelor's degree in Electronics and Information Engineering. She is now a senior researcher in Norwegian Research Center (NORCE). At the same time, she is a scientific researcher at Centre of Artificial Intelligence Research (CAIR) in UiA. She is currently serving as a Board Member of Norwegian Association for Image Processing and Machine Learning. Her research interests include Learning Automata, Mathematical Analysis on Learning Algorithms, Deep Learning, Natural Language Processing and Computer Vision.

Anis Yazidi received the M.Sc. and Ph.D. degrees from the University of Agder, Grimstad, Norway, in 2008 and 2012, respectively. He was a Researcher with Teknova AS, Grimstad. From 2014 to 2019, he was an Associate Professor with the Department of Computer Science, Oslo Metropolitan University, Oslo, Norway, where he is currently a Full Professor, leading the research group in applied artificial intelligence. He is also a Professor II with the Norwegian University of Science and Technology (NTNU), Trondheim, Norway. His current research interests include Machine Learning, Learning Automata, Stochastic Optimization, and Autonomous Computing.

Dr. John Oommen was born in Coonoor, India on September 9, 1953. He obtained his B.Tech. degree from the Indian Institute of Technology, Madras, India in 1975. He obtained his M.E. from the Indian Institute of Science in Bangalore, India in 1977. He then went on for his M.S. and Ph. D. which he obtained from Purdue University, in West Lafayettte, Indiana in 1979 and 1982 respectively. He joined the School of Computer Science at Carleton University in Ottawa, Canada, in the 1981-82 academic year. He is still at Carleton and holds the rank of a Full Professor. Since July 2006, he has been awarded the honorary rank of Chancellor's Professor, which is a lifetime award from Carleton University. His research interests include Automata Learning, Adaptive Data Structures, Statistical and Syntactic Pattern Recognition, Stochastic Algorithms and Partitioning Algorithms. He is the author of more than 495 refereed journal and conference publications, and is a Life Fellow of the IEEE and a Fellow of the IAPR. Dr. Oommen has also served on the Editorial Board of the IEEE Transactions on Systems, Man and Cybernetics, and Pattern Recognition.

Appendix C

The ADE HDPA Papers

C.1 Enhancing the Speed of Hierarchical Learning Automata by Ordering the Actions – A Pioneering Approach

R. O. Omslandseter, L. Jiao, and J. B. Oommen, "Enhancing the Speed of Hierarchical Learning Automata by Ordering the Actions – A Pioneering Approach," *AI 2022: Advances in Artificial Intelligence, AI 2022 (AJCAI 2022)*, Lecture Notes in Computer Science, vol 13728, pp. 775–788, Springer International Publishing, December 2022.
DOI: https://doi.org/10.1007/978-3-031-22695-3_54

Enhancing the Speed of Hierarchical Learning Automata by Ordering the Actions – A Pioneering Approach

Rebekka Olsson Omslandseter[1], Lei Jiao[1], and B. John Oommen[2]

[1] Dept. of Information and Communication Technology,
University of Agder, 4879, Grimstad, Norway,
{rebekka.o.omslandseter, lei.jiao}@uia.no
[2] School of Computer Science,
Carleton University, K1S 5B6, Ottawa, Canada

Abstract. For the past six decades, the operation of Learning Automata (LA) has involved states and action probabilities. These have been central to "remembering" the quality of the actions chosen during the learning. The latest enhancements have also incorporated estimates of the actions' reward probabilities. However, a phenomenon that has never been used to-date is that of considering how these actions *themselves*, can be *ordered*. Ordering the actions in traditional LA is rather meaningless unless one resorts to invoking the theory of Random Races [1]. However, we show that such an ordering makes sense if the automata operate hierarchically, within a tree, with the actions being placed at the leaves. In this paper, we shall show that when the LA are arranged "in a tree formation", and when the learning is achieved within such a tree, the hierarchical LA has a superior performance if the actions located at the leaves of the tree are arranged suitably. While this concept can be incorporated in *any* hierarchical LA, we demonstrate its power for the most recent machine, i.e., the Hierarchical Discretized Pursuit Automaton (HDPA). These strategies can also be included in the Hierarchical Continuous Pursuit Automaton (HCPA), and to both of these which utilize traditional Maximum Likelihood (ML) or Bayesian estimates [2]. The experimental results presented here are very impressive, and so, if we consider the chronology of LA from FSSA through VSSA, the Estimator schemes, and the recent hierarchical LA, our modest claim is that the inclusion of the ADE represents the state-of-the-art which is not easily surpassed.

Keywords: Learning Automata · Reinforcement Learning · Hierarchical Learning Automata.

1 Introduction

The field of Learning Automata (LA)[3] concerns non-human agents learning the optimal action from a set of actions through the principles of Reinforcement Learning (RL). The learning agent in LA is often referred to as the *Learning Automaton*. The system that the learning agent operates in and learns through interactions with, is often referred to as the *Environment*. The learning agent selects an action, and the Environment responds with a feedback to the automaton. The feedback can be a set of discrete responses or from a continuous range, but most commonly, the feedback is binary, consisting of either a reward or a penalty. Depending on the LA's learning policy, the LA updates its knowledge based on the feedback, and hopefully, learns to output the action that yields the highest probability of getting a reward over time. The LA learns in a *semi-supervised* manner, meaning that it does not need examples of solutions but learns via the feedback in a trial-and-error mode. The metric of quantifying the performance of LA is the average number of *iterations* it takes, over an ensemble of experiments. For the Variable Structure Stochastic Automata (VSSA) type of LA used in this paper, the algorithm converges once the LA attains a specific probability level that is arbitrarily close to unity.

1.1 Memory Considerations

Learning cannot be achieved without remembering certain quantities during the process. We shall informally refer to these as the "*memory*". However, even if one remembers all the pertinent information in a very efficient manner, the learning will not succeed if the algorithm that utilizes the memory, is poor. To bring out the salient features of the pioneering contribution of this paper, we briefly itemize the respective components of the different families.

- **FSSA**: Fixed Structure Stochastic Automata (FSSA) are LA, where the memory is encapsulated in states which are identical to those possessed by Finite State Machines or flip flops. Examples of these are the Tsetlin, Krinsky, and Krylov LA [3]. In each case, the learning algorithm directs the LA to move across the states based on the response from the Environment, and each of the latter boast their own individual strategy. Correspondingly, they all have different convergence characteristics.
- **VSSA**: Unlike FSSA, in VSSA, the memory is contained in the action probability vector, $P(n)$. The action is chosen based on $P(n)$, which is then communicated to the Environment. $P(n+1)$ is obtained in the next step, and is based on $P(n)$, the action chosen, $\alpha(n)$, and the feedback that the Environment provides, $\beta(n)$. The updating algorithm, on the other hand, can be varied and includes, among others, the Linear Reward-Penalty (L_{R-P}) scheme, the Linear Reward-Inaction (L_{R-I}) scheme, the Linear Inaction-Penalty (L_{I-P}) scheme, and the Linear Reward-ϵPenalty ($L_{R-\epsilon P}$) [4], [5].

[3] The term LA is used interchangeably to address the field of Learning Automata or the Learning Automata themselves, depending on the context.

- **Estimator Algorithms**: In Estimator LA, the memory resides in the action probability vector and running estimates of the reward probabilities. In the Pursuit algorithm, the learning algorithm now increases the probability of the currently recorded best action and not of the action that is chosen. In the Pursuit algorithm, only the probability of the best action is increased. In the Generalized Pursuit, the action probabilities of a subset of actions are increased, while the rest of the probabilities are decreased. One has to mention that the estimates can be done in an ML or Bayesian manner [2], and the updates done in a continuous or a discretized paradigm.
- **Hierarchical LA**: In the family of hierarchical LA, the machines are arranged in a tree structure, and the actions are at the leaves. These individual LAs can be VSSA or can be Pursuit machines themselves. More details of this are included in the next section.

The above bullets briefly encapsulates the entire prior art.

1.2 Action Ordering Considerations

The reader will observe that throughout the above discussions, the ordering of the actions has remained insignificant. This is valid because the ordering is unknown unless one resorts to a prior Random Race competition that is not relevant to our present study [1]. Of course, the ordering of the actions in an action probability vector is also meaningless.

The hypothesis of this study is that there is an advantage to ordering the actions. Clearly, such an ordering can only be enforced if there are crude estimates of the reward probabilities. If they are arranged linearly, ordering the actions can enhance the corresponding choice by resorting to a fast searching mechanism, as opposed to a linear search. We shall not elaborate on that issue here.

However, let us consider the case when the automata operate in a hierarchical manner. The actions then are placed at the leaves of the tree, and the decisions of the individual LA trickle up to the root. Our hypothesis is that rather than keeping the leaves completely unordered, information gleaned during the initial learning phase can be used to order them, and to yield a superior performance. This is precisely the hypothesis and contribution of our paper.

1.3 Contributions of this Paper

The contributions of this paper can be summarized as follows:

- Unlike the prior art, we show that there is an advantage in considering the ordering of the actions when the LA operate in a hierarchical manner.
- We demonstrate this, by considering the most recent machine in the field, the HDPA.
- We confirm the hypothesis, by reporting the results of extensive simulations in different Environments and a host of distributions.

As mentioned above, the concept presented in this paper is true, not only for the HDPA, but also for any other type of machine utilizing a hierarchical structure.

2 Related Work

The paradigm of LA originated in the 1960s with Michael Lvovitch Tsetlin and his innovation of learning agents and, ultimately, the Tsetlin Automata [6]. Later advancements followed, and the types of LA are generalized into two categories, namely FSSA and the VSSA. In VSSA, we have the Linear Reward-Penalty(L_{R-P}) scheme, the Linear Reward-Inaction(L_{R-I}) scheme, the Linear Inaction-Penalty(L_{I-P}) scheme, and the Linear Reward-ϵPenalty($L_{R-\epsilon P}$) [4], [5]. In these different schemes, the probability vector is updated linearly. The updating can also be done in a non-linear manner [4], [5], [7]. At the same time, VSSA schemes can be continuous or discrete [8]. Continuous type VSSA updates the probability in a multiplicative manner with a factor, while the discretized type updates the probability with a constant in each update. Due to the multiplicative updating, the continuous type can experience slower algorithm speeds than the discretized type. Thus, when an action selection probability gets closer to unity, the change in its probability becomes less and less. The continuous and discrete updating functionality has been investigated mathematically in [9], [10].

Another major discovery in the field of LA was the Estimator-based Algorithms (EAs), which significantly increased the convergence speed of VSSA [11]. The concept of EAs is the utilization of estimation. In more detail, the LA keeps reward estimates while in operation, using these estimates to pursue the currently most promising action (referred to as *Pursuit* in the Literature) [12]. Researchers combined the Pursuit concept with discretized updating, leading to the paradigm of Discrete Estimator Algorithms (DEAs) [13], superior to earlier VSSA variants in terms of convergence speed.

Although all of the advances mentioned above increased the applicability and efficiency of LA dramatically, VSSA still had an impediment as the number of actions (possible solutions to a problem) became *large* (e.g., more than ten [14]). Therefore, the authors of [14] proposed the HCPA, bringing structure to the domain of VSSA. The HCPA was a quantum step to the field of LA, making VSSA able to handle a large number of actions. However, as the accuracy requirement to the HCPA became large, e.g., above 0.98, the HCPA suffered from its multiplicative property of updating its action selection probabilities, resulting in sluggish convergence. In [15], the HDPA was proposed, combining VSSA, discretized updating, the Pursuit concept, and structure. The HDPA provides a solution to problems with many actions and high accuracy requirements, constituting the state-of-the-art for generic LA, being significantly faster when compared to the HCPA.

3 Incorporating Ordering into an Hierarchical LA

Although the HDPA improved the convergence speed significantly for high accuracy requirements, we experienced that the convergence speeds were substantially dependent on the action distribution on the leaf level of the tree. Linked to the concept of Random Races [1], where the LA is modeled to find an ordering of

the actions in an ascending/descending order, we hypothesize that we can order the actions in a manner that is beneficial to the algorithm. More specifically, we propose that we can use an Estimation Phase in the HDPA process and also the estimated reward probabilities to reorder the actions at the leaf level to yield an improved performance. Consequently, in this paper, we propose the Action Distribution Enhancing (ADE) approach for enhancing the convergence speed of the HCPA and HDPA. The improvement in the convergence speed becomes more noticeable as the number of actions at the leaf level increases. Therefore, organizing the actions in an improved manner can significantly reduce the number of iterations before the convergence is achieved. While this was our intended hypothesis, as demonstrated through extensive simulations documented later in the paper, we show that the ADE approach is, indeed, beneficial compared to randomly initializing the actions at the leaf level of the tree. The reader should note that the problems that LA can solve are random in nature, and for a real problem, no information about the reward probabilities can be known *a priori*. Therefore, understandably, the Estimation Phase is needed to obtain an improved ordering.

3.1 Motivating arguments

To motivate the development of our new paradigm, we consider a problem involving four actions $\mathcal{A} = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ with the corresponding estimated action probability vector $\hat{D} = \{\hat{d}_1, \hat{d}_2, \hat{d}_3, \hat{d}_4\}$, taken over an initial estimation phase of 20 iterations. The reader must please observe that because the number of iterations are small, the corresponding estimates will be inaccurate. Also, before we proceed with the arguments, it is wise to see how these estimates will effect the learning process. Further, in the interest of simplicity, we proceed with the discussion by considering the case of the HDPA instead of any other arbitrary hierarchical LA.

At the leaf levels, the four actions are to be placed in one of the 4! positions. It is also obvious that the descending and ascending orders of the placements of the actions are merely mirror reflections of each other. On a deeper examination of an Hierarchical Pursuit LA, one observes that from every level, the estimates of the most suitable actions chosen at that level will be trickled up. This implies that the automata at each level will be dealing with problems of different complexities. However, the most important automaton is the one placed at the root, because that governs, or rather dictates, the operations of all the automata below it.

Consider the following figure in which the four actions are placed at the leaves in the descending order (Fig. 1a). From Fig. 1a, we see that the LA $A_{1,1}$ has to deal with actions whose reward estimates are $\frac{15}{20}$ and $\frac{11}{20}$. Similarly, when the actions are in a more random order, Fig. 1b, the corresponding reward estimates for the LA $A_{1,1}$ are $\frac{15}{20}$ and $\frac{3}{20}$. If all goes well as in a perfect world, the trickled up estimates are those of the optimal actions of their corresponding children. The root level, which is encountering the most important task, has now to deal with distinguishing between the reward estimates $\frac{15}{20}$ and $\frac{7}{20}$ in Fig. 1a, and $\frac{15}{20}$ and $\frac{11}{20}$ in Fig. 1b. This means that the root automaton, that has to solve the

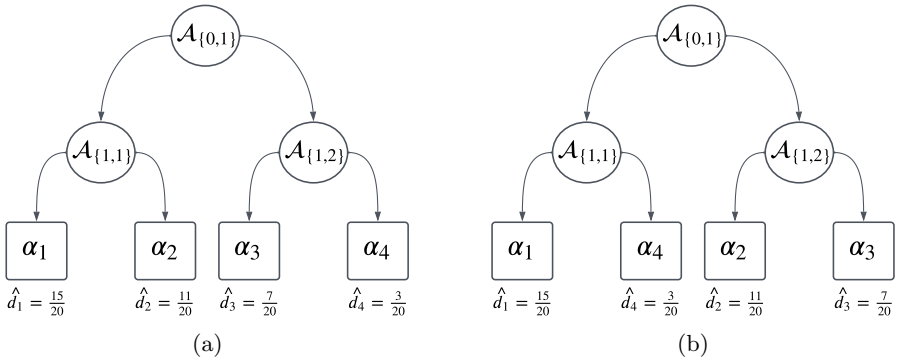


Fig. 1: Example of two hierarchical tree structures for four actions with different action distributions at the leaf level.

most discriminating problem of all the automata, has to resolve actions α_1 and α_2, in the case of Fig. 1b, which is much more difficult than the problem in Fig. 1a.

When we use the euphemistic expression above "if all goes well in a perfect world", we emphasize that it is statistically not at all unrealistic. This is because by the law of large numbers or the Estimation Phase in the Bayesian case, the estimates of the reward probabilities will converge to their true values with an arbitrarily high accuracy. Thus, the asymptotic arguments (and probabilities) of the trees of Fig. 1a and Fig. 1b will still be valid[4].

4 The Action Distribution Enhancing (ADE) Approach

As mentioned earlier, such an ADE approach applies to both the HCPA and the HDPA, and indeed, to any hierarchical LA. However, because the HDPA has demonstrated superior performance to the HCPA for high accuracy requirements, and we have limited space, we only highlight the ADE approach for the HDPA, by understanding that the principles for the HCPA are analogous. The ADE approach for the HCPA is similar to the approach explained for the HDPA.

The ADE approach concerns distributing the actions at the leaf level of the hierarchical tree in an improved manner. For a practical, real-life problem, there can be little information as *a priori* information about the actions. This is why the distribution of the actions at the leaf level is an intricate problem which has not yet been considered in the Literature. The ADE approach is two-pronged. The first prong is the *Estimation Phase*, used for estimating the action reward probabilities. The second concerns distributing the actions in an improved manner, and is referred to as the *Reallocation Process*.

[4] The proof that the ADE approach represents a superior solution compared with unordered solutions, will be proven in the extended version of the paper [16].

The first part of the ADE approach concerns the Estimation Phase. In the HDPA, the estimated reward probability and the action selection probabilities of LAs throughout the tree structure are initialized as 0.5, according to [14] and [15]. Thus, the HDPA estimates the reward probabilities as per the Pursuit concept (maintaining an estimated reward probability vector). We propose a standalone Estimation Phase with the ADE approach prior to the HDPA starting its regular operation. This means that in this phase, we include θ iterations per action for estimating their reward probabilities. These estimates are further utilized as to initialize the corresponding values in the regular operation of the HDPA, which happens after the Reallocation Process.

The second part of the ADE approach concerns the Reallocation Process, which distributes the actions in an improved manner. For a two-action LA configured tree, such an organization can be achieved by ordering the actions according to their estimated reward probabilities in either an ascending or descending order. In this way, asymptotically, the optimal and second optimal actions will share the same LA at the level below the root. Thus, asymptotically, the algorithm will have achieved a correct estimation of the reward probabilities, and the actions will be distributed in an improved manner. Clearly, due to the stochastic nature of the problem, the Estimation Phase might not always yield a perfect interpretation of the reward estimates and the corresponding trees. This phenomenon and its consequences are explained further in the section with the experimental results (Section 5).

The Reallocation Process uses the estimated reward probabilities from the Estimation Phase to reallocate the actions to the tree's leaves in an ascending/descending order. Thus, after the Estimation Phase, the actions are given a new location at the leaf level. The reader should note that the estimates also need to be updated according to this new ordering. By maintaining these estimates, the information from the Estimation Phase is also retained. After the Reallocation Process, the HDPA starts its *normal* operation, by utilizing the reward estimates from the Estimation Phase[5].

5 Experimental Results

To demonstrate the effectiveness of the proposed ADE approach presented in this paper, we conducted experiments with various action distributions, numbers of actions, and Environments. For quantifying the effectiveness of the LA algorithms, we recorded the number of iterations required before convergence, as this is the most common evaluating measurement [5]. As mentioned earlier, the convergence requirement for VSSA is that one of the action selection probabilities attains a certain threshold (T). In our experiments, we tested different convergence criteria, and report the results for the most utilized convergence threshold to be 0.995. The reader should note that this metric is helpful because

[5] Although the algorithm have been explained in details verbally in this paper, a more detailed programmatic description of the algorithm will be presented in an extended version of the paper [16].

it will remain identical, regardless of the computing power on the machine used for the experiments, or the efficiency of the code or language used[6].

Paired together with this, is the accuracy of the algorithm. Due to the stochastic nature of the problem, one often conducts more experiments and reports the average performance. In terms of accuracy, we usually measure this as the percentage of experiments which have converged to the optimal action. Consequently, when a 100% accuracy is achieved, all the experiments conducted in an ensemble of experiments have converged to the optimal action (i.e, the action with the highest reward probability). The reader should note that the number of iterations used for the Estimation Phase is also reckoned into the overall number of iterations in our experimental results.

In LA, the tuning of the learning parameter leads to a trade-off between the accuracy and the speed. Generally, as the learning parameter becomes smaller, the number of iterations increases, and at the same time, the algorithm performs more accurately. The same dilemma applies to the algorithm proposed in this paper. As demonstrated in the experiments, placing the optimal and sub-optimal actions in opposite parts of the tree at the leaf level requires more iterations than establishing them as entities in the same part of the tree (for example, i.e., with an ascending/descending ordering). Thus, the ascending/descending orders generally require substantially less number of iterations before convergence.

With our experiments, we wanted to demonstrate the behavior of the HDPA for different action distributions at the leaf level, thereby demonstrating the improved performance with the ADE approach. In practice, we have little or no *a priori* information about the reward estimates in a real-life scenario. In our simulations, we know that with a knowledge of the exact reward probabilities, we are able to execute the programs for different action distributions with and without the ADE approach. By demonstrating the phenomena for different configurations, we intend to highlight the improved performance that can be achieved in a real-life scenario by using the ADE approach with the Estimation Phase and Reallocation Process before the actual LA learning is performed.

In our simulations, we denote the *real* reward probabilities, i.e., the probability of getting a reward for selecting a certain action, as d_j, where $j \in \{1, 2, ..., 2^K\}$. If nothing else is specified, for the ADE HDPA, we used a descending ordering in the Reallocation Process. To help understand the ordering at the leaf level, we present visualizations of the action distributions with their corresponding reward probabilities. Please note that these visualizations show the real reward probability of α_1 to α_{2^K}, i.e., d_1 to d_{2^K}, for the actions at the leaf level. Consequently, the HDPA without the ADE will run the experiments

[6] In our experiments, we have configured the convergence criterion as being achieved once *any* of the LA has attained a certain threshold of choosing one of the actions in its action probability vector. However, in [15], they defined the convergence as being achieved only when all the LA along the path to a leaf action had attained the prescribed threshold. Thus, the convergence criterion in this paper is different, i.e., it utilizes the "logical or" instead of the "logical and", making the algorithms (i.e., both the HDPA without/with the ADE) attain a faster convergence.

with the actions ordered as displayed by the configuration. Conversely, the ADE HDPA will reallocate the actions at the leaf level in an ascending/descending order based on the corresponding reward estimates.

5.1 Simulation 1: 8 Actions

Let us first consider the results for Simulation 1 presented in Tables 1 and 2, which involve Environments of 8 actions. The difference between the two tables is that Table 1 does not incorporate the ADE, and Table 2 does. The categoric superiority of the results in Table 2 demonstrates the power of the ADE.

Table 1: Experimental results for different action distributions without the ADE approach for eight actions with $T = 0.995$ as the convergence criterion. The results were averaged over $1,000$ experiments, with $\Delta = 9e{-}5$.

Config.	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	Avg	Std	Acc.
1	0.99	0.95	0.87	0.6	0.43	0.54	0.67	0.51	6,727.40	42.76	100%
2	0.87	0.6	0.43	0.54	0.67	0.51	0.99	0.95	6,791.75	56.81	100%
3	0.87	0.6	0.99	0.95	0.43	0.54	0.67	0.51	6,730.42	42.03	100%
4	0.87	0.6	0.43	0.99	0.95	0.54	0.67	0.51	7,082.38	215.46	100%
5	0.99	0.6	0.43	0.54	0.67	0.51	0.87	0.95	7,109.98	220.78	100%
6	0.95	0.99	0.51	0.67	0.54	0.43	0.6	0.87	6,791.15	57.52	100%
7	0.99	0.95	0.87	0.67	0.6	0.54	0.51	0.43	6,713.85	42.49	100%

Table 2: Experimental results for different action distributions with the ADE HDPA with eight actions and $T = 0.995$ as the convergence criterion. The results were averaged over $1,000$ experiments, with $\Delta = 9e{-}5$.

Config.	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	Avg	Std	Acc.
1	0.99	0.95	0.87	0.6	0.43	0.54	0.67	0.51	6,810.10	44.66	100%
2	0.87	0.6	0.43	0.54	0.67	0.51	0.99	0.95	6,824.48	51.26	100%
3	0.87	0.6	0.99	0.95	0.43	0.54	0.67	0.51	6,823.52	49.19	100%
4	0.87	0.6	0.43	0.99	0.95	0.54	0.67	0.51	6,821.79	49.51	100%
5	0.99	0.6	0.43	0.54	0.67	0.51	0.87	0.95	6,825.32	49.49	100%
6	0.95	0.99	0.51	0.67	0.54	0.43	0.6	0.87	6,813.32	47.15	100%
7	0.99	0.95	0.87	0.67	0.6	0.54	0.51	0.43	6,810.91	44.67	100%

In Simulation 1, we ran 1,000 experiments with similar reward probabilities associated with the Environment, but with different action distributions. In Tables 1 and 2 we have marked the optimal and optimal action in at the leaf level with a different background color. For example, we can observe Config. 5, where the optimal and sub-optimal actions are located in opposite parts of the tree. We had earlier explained that this configuration was the hardest for the HDPA

to handle. As opposed to this, having the actions with regard to the reward probabilities in an ascending/descending order, like the case of Config. 7, would be easier. This is confirmed from the results in Table 1, where the HDPA with the ADE has a superior performance. Config. 4 and Config. 5 required the highest number of iterations and had high standard deviations (Std). However, for Config. 7, where the actions were ordered in a descending order according to the reward probabilities, the number of iterations was the lowest. This is obvious since this is the most optimal setting. Again, the results for the ascending and descending orders are almost identical.

In Table 2 we present the results for the experiments for the ADE HDPA for Simulation 1. As we observe from the table, some configurations required more iterations than the other configurations without the ADE approach. However, the ADE HDPA was more consistent in the number of iterations, and had a more stable and smaller standard deviation. Config. 2, which is the reversed form of Config. 6, yielded a little higher standard deviation than the others. We also, tested Config. 2 for the ADE HDPA with an ascending ordering in the Reallocation Process, and these experimental results were similar to those of Config. 6. More specifically, for 1,000 experiments with the ADE HDPA and an ascending ordering in its Reallocation Process, the algorithm required 6,818.14 iterations on average and a standard deviation of 48.46 yielded 100% accuracy.

In real-life, we do not know the underlying reward probabilities. Therefore, we can only tune the number of tests, θ, that is used in the Estimation Phase. Testing each action for a larger number of iterations in the Estimation Phase will make the ADE more certain that it has estimated the reward probabilities correctly, and it will, thus, order them correctly as well. However, in most cases, because we want a fast convergence, a rough estimate might be sufficient. In Simulation 1, we used $\theta = 12$. If we had *perfect* estimation of the reward probabilities, we would have similar results to Config. 7 without the ADE.

5.2 Simulation 2: 16 Actions

In Simulation 2, we increased the number of actions to 16. Again, we set the value of θ to be 12. The different original action distributions are visualized in Fig. 2, where we focus on the optimal and sub-optimal actions' locations in the different configurations. However, the actions in between them were distributed more or less *randomly*. In subsequent simulations, we shall highlight what happens when we have more constructed forms in the original distributions.

In Tables 3 and 4, we present the results for Simulation 2. Analogous to Simulation 1, we see that the number of iterations is more consistent, and that the standard deviation is smaller for the ADE HDPA (Table 4). The HDPA without ADE still had better results for the configurations where the actions were manually ordered in an ascending/descending order, which is quite understandable. Thus, the ADE HDPA did probably not achieve the *perfect* estimation of the reward probabilities, since the number of iterations used for the estimation, where $\theta = 12$ and $R = 16$ ($\theta R = 192$), was too small.

Furthermore, the simulation demonstrated a bigger gain using the ADE approach for 16 actions when compared with the 8 actions case. As an example, for Row No. 1, the ADE HDPA used approximately 11,550 iterations before convergence, while the HDPA without the ADE used approximately 12,700 iterations, which yielded a superiority of more than 1,000 iterations. Thus, the ADE had an approximately 9.95% better performance in terms of the number of iterations. In comparison, for Simulation 1, the biggest gain of using the ADE approach was approximately 4.25%.

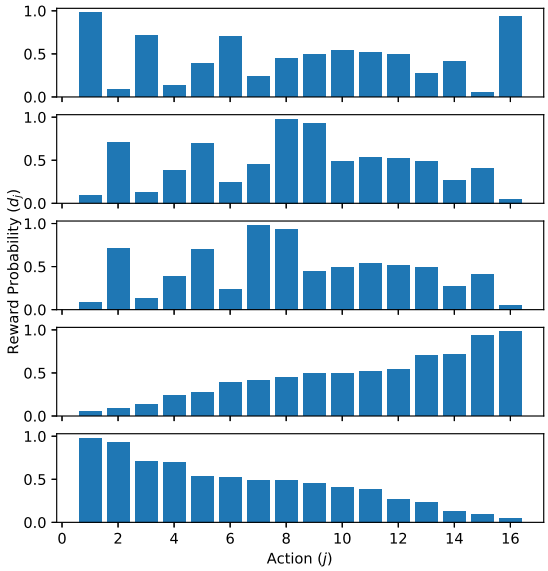


Fig. 2: The figure shows the action distribution in accordance with the reward probabilities for Simulation 2. Config. 1 is at the top, Config. 5 is at the bottom, and the others are ordered in between them systematically.

5.3 Simulation 3: 32 Actions

In Tables 5 and 6, we present the results obtained for Simulation 3, which involved 32 actions. The configurations in these experiments followed the same concept as depicted in Fig. 2 for Simulation 2. We can observe that the HDPA without the ADE required considerably more iterations for the case when the optimal and sub-optimal actions were in opposite parts of the tree (Row No. 1),

Table 3: Experimental results for different action distributions without the ADE for Simulation 2, with 16 actions and 0.995 as the convergence criterion. The results were averaged over 100 experiments, with $\Delta = 6.75e{-}5$. The different rows represent different action configurations as described in the second column.

Row No.	Configuration characteristics	Avg	Std	Acc.
1	Config 1 in Fig.2	12,691.92	509.64	100%
2	Config 2 in Fig.2	12,362.99	413.44	100%
3	Config 3 in Fig.2	11,674.32	100.95	100%
4	Ascending	11,285.18	83.24	100%
5	Descending	11,291.90	86.09	100%

Table 4: Experimental results for different action distributions with the ADE for Simulation 2, with 16 actions and 0.995 as the convergence criterion. The results were averaged over 100 experiments, with $\Delta = 6.75e{-}5$. The different rows represent different action configurations as described in the second column.

Row No.	Configuration characteristics	Avg	Std	Acc.
1	Config 1 in Fig.2	11,556.77	111.29	100%
2	Config 2 in Fig.2	11,540.57	106.94	100%
3	Config 3 in Fig.2	11,543.75	94.47	100%
4	Ascending	11,573.89	119.07	100%
5	Descending	11,520.16	91.70	100%

i.e., compared with having the actions ordered in a descending order (Row No. 5). The numbers of iterations were approximately $17,700$ and $15,800$, respectively. Comparing the results in Table 5 with those in Table 6, we observe that the ADE HDPA had a more consistent performance in terms of the number of iterations and the standard deviations. For Row No. 1 in the tables, we see that the HDPA with the ADE required approximately $16,350$, while the HDPA without the ADE required $17,700$, which is approximately 8.26% worse.

Table 5: Experimental results for different action distributions without the ADE for Environments with 32 actions where 0.995 is the convergence criterion. The results were averaged over 100 experiments, with $\Delta = 4.5e{-}5$. The different rows represent different configurations as described in the second column.

Row No.	Configuration characteristics	Avg	Std	Acc.
1	As in Config 1 in Fig.2	17,690.81	849.65	100%
2	As in Config 2 in Fig.2	17,980.64	742.41	100%
3	As in Config 3 in Fig.2	16,911.9	600.69	100%
4	Ascending	15,807.09	89.47	100%
5	Descending	15,806.49	90.23	100%

Table 6: Experimental results for different action distributions with the ADE HDPA for 32 actions and 0.995 as the convergence criterion. The results were averaged over 100 experiments, with $\Delta = 4.5e{-}5$ and $\theta = 12$. The different rows represent different configurations as described in the second column.

Row No.	Configuration characteristics	Avg	Std	Acc.
1	As in Config 1 in Fig.2	16,338.12	114.54	100%
2	As in Config 2 in Fig.2	16,302.25	121.44	100%
3	As in Config 3 in Fig.2	16,327.96	121.10	100%
4	Ascending	16,371.00	134.76	100%
5	Descending	16,256.67	107.27	100%

6 Conclusion

In this paper we have proposed the novel Action Distribution Enhancing (ADE) approach for optimally configuring the underlying hierarchical tree representing the distribution of the actions in the HCPA/HDPA. The ADE involves two phases, the first of which estimates the action probabilities very crudely, and subsequently assigns the actions at the leaves of the tree. The corresponding LA then operate in a hierarchical manner, each of them involving two actions. Our hypothesis was that if the leaves were arranged in an ascending/descending order, the collection of hierarchical automata would perform in their most optimized manner, and we confirmed this hypothesis by both a formal theoretical analysis and experimentally [16].

We have then proceeded to verify the power of incorporating the ADE into problems involving different numbers of automata, and various Environments with corresponding reward probabilities. Quite briefly stated, our simulation results uniformly confirm that the inclusion of the ADE significantly stabilizes and increases[7] the convergence speed of the hierarchical machine.

If we consider the chronology of LA from its infancy in FSSA through VSSA, the Estimator approaches, and the more-recent hierarchical schemes, we modestly believe that the inclusion of the ADE represents the state-of-the-art which will not be able to be surpassed too easily.

References

1. D. T. H. Ng, B. J. Oommen and E. R. Hansen, "Adaptive Learning Mechanisms for Ordering Actions Using Random Races," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 5, pp. 1450-1465, 1993.
2. X. Zhang, O.-C. Granmo and B. J. Oommen, "On Incorporating the Paradigms of Discretization and Bayesian Estimation to Create a New Family of Pursuit Learning Automata," *Applied Intelligence*, vol. 39, no. 4, pp. 782-792, 2013.

[7] The speed of HDPA with ADE, compared with vanilla HDPA, indeed decreases a bit for the ascending/descending case. Nevertheless, considering the significant speed gain for other cases, the average speed increases.

3. M. L. Tsetlin, *Automaton Theory and the Modeling of Biological Systems*, New York: Academic Press, 1973.
4. S. Lakshmivarahan, *Learning Algorithms Theory and Applications*, ed. 1, New York: Springer-Verlag, 1981.
5. K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*, Dover Books on Electrical Engineering Series, Dover Publications, Courier Corporation, 2013.
6. M. L. Tsetlin, "Finite Automata and Modeling the Simplest Forms of Behavior," *Uspekhi Matem Nauk*, vol. 8, no.4, pp. 1–26, 1963.
7. S. Lakshmivarahan and M. A. L. Thathachar, "Absolutely Expedient Learning Algorithms for Stochastic Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 3, pp. 281–286, 1973.
8. B. J. Oommen, "Absorbing and Ergodic Discretized Two-Action Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 2, pp. 282–293, 1986.
9. B. J. 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.
10. X. Zhang, O.-C. Granmo, B. J. Oommen, "Discretized Bayesian Pursuit - A New Scheme for Reinforcement Learning," *Advanced Research in Applied Artificial Intelligence*, vol. 7345, pp. 784–793, 2012.
11. M. A. L. Thathachar and P. S. Sastry, "Estimator Algorithms for Learning Automata," *Proceedings of the Platinum Jubilee Conference on Systems and Signal Processing*, Department of Electrical Engineering, Indian Institute of Science, 1986.
12. X. Zhang, B. J. Oommen and O.-C. Granmo, "The Design of Absorbing Bayesian Pursuit Algorithms and the Formal Analyses of Their ϵ-Optimality," *Pattern Analysis and Applications*, vol. 20, pp. 797–808, 2017.
13. J. K. Lanctot and B. J. Oommen, "Discretized Estimator Learning Automata," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1473–1483, 1992.
14. A. Yazidi, X. Zhang, L. Jiao, and B. J. 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*, vol. 31, no. 2, pp. 512–526, 2020.
15. R. O. Omslandseter, L. Jiao, X. Zhang, A. Yazidi, and B. J. Oommen, "The Hierarchical Discrete Learning Automaton Suitable for Environments with Many Actions and High Accuracy Requirements," *in Proc. AJCAI 2022. Lecture Notes in Computer Science*, vol. 13151, pp. 507–518, 2022.
16. R. O. Omslandseter, L. Jiao, and B. J. Oommen, "Pioneering Approaches for Enhancing the Speed of Hierarchical LA by Ordering the Actions". Unabridged version of this paper. To be submitted for publication.

Appendix D

The NOMA Papers

D.1 User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution

This paper has been published as:

R. O. Omslandseter, L. Jiao, Y. Liu, and J. B. Oommen, "User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution," *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices, IEA/AIE 2020*, vol 12144, pp. 299–311, Springer International Publishing, September 2020.
DOI: https://doi.org/10.1007/978-3-030-55789-8_27

User Grouping and Power Allocation in NOMA Systems: A Reinforcement Learning-Based Solution

Rebekka Olsson Omslandseter[1], Lei Jiao[1], Yuanwei Liu[2], and B. John Oommen[1,3]

[1] University of Agder, Grimstad, Norway {rebekka.o.omslandseter,lei.jiao}@uia.no
[2] Queen Mary University of London, London, UK yuanwei.liu@qmul.ac.uk
[3] Carleton University, Ottawa, Canada oommen@scs.carleton.ca

Abstract. In this paper, we present a pioneering solution to the problem of user grouping and power allocation in Non-Orthogonal Multiple Access (NOMA) systems. There are two fundamentally salient and difficult issues associated with NOMA systems. The first involves the task of grouping users together into the pre-specified time slots. The subsequent second phase augments this with the solution of determining how much power should be allocated to the respective users. We resolve this with the first reported Reinforcement Learning (RL)-based solution, which attempts to solve the partitioning phase of this issue. In particular, we invoke the Object Migration Automata (OMA) and one of its variants to resolve the user grouping problem for NOMA systems in stochastic environments. Thereafter, we use the consequent groupings to infer the power allocation based on a greedy heuristic. Our simulation results confirm that our solution is able to resolve the issue accurately, and in a very time-efficient manner.

Keywords: Learning Automata · Non-Orthogonal Multiple Access · Object Migration Automata · Object Partitioning

1 Introduction

The Non-Orthogonal Multiple Access (NOMA) paradigm has been proposed and promoted as a promising technique to meet the future requirements of wireless capacity [6]. With NOMA, the diversity of the users' channels and power is exploited through Successive Interference Cancellation (SIC) techniques in receivers [1]. This technology introduces questions concerning users who are ideally supposed to be grouped together, so as to obtain the maximum capacity gain. Additionally, the power level of the signal intended for each user is a crucial component for the successful SIC in NOMA operations. Consequently, it is an accepted fact that the performance of NOMA is highly dependent on both the grouping of the users and the power allocation.

The user grouping and power allocation problems in NOMA systems are, in general, intricate. First of all, the user grouping problem, in and of itself, introduces a combinatorially difficult task, and is infeasible as the number of users increases. This is further complicated by the channel conditions, and the random nature of the users' behaviors in communication scenarios. For this reason, the foundation for grouping, and consequently, for power allocation, can change rapidly. It is, therefore, necessary for a modern communication system to accommodate and adapt to such changes.

The user grouping in NOMA systems, is akin to a classic problem, i.e., to the Object Partitioning Problem (OPP). The OPP concerns grouping "objects" into sub-collections, with the goal of optimizing a related objective function, so as to obtain an optimal grouping [3]. Our goal is utilize Machine Learning (ML) techniques to solve this, and in particular, the ever-increasing domain of Reinforcement Learning (RL), and its sub-domain, of Learning Automata (LA). When it concerns RL-based solutions for the OPP, the literature reports many recent studies to solve Equi-Partitioning Problems (EPPs). EPPs are a sub-class of the OPP, where all the groups are constrained to be of equal size. Among these ML solutions, the Enhanced Object Migration Automata (EOMA) performs well for solving different variants of EPPs [2]. They can effectively handle the stochastic behavior of the users, and are thus powerful in highly dynamic environments, similar to those encountered in the user grouping phase in NOMA systems.

Moving now to the second phase, the task of allocating power to the different users of a group in NOMA systems, further complicates the NOMA operation. However, a crucial observation is that the problem resembles a similar well-known problem in combinatorial optimization, i.e., the *Knapsack Problem* (KP). KPs, and their variants, have been studied for decades [4], and numerous solutions to such problems have been proposed. Among the numerous solutions, it is well known that many fundamental issues can be resolved by invoking a greedy solution to the KP. This is because a greedy solution can be exquisite to a highly complex problem, and can quickly utilize a relation among the items, to yield a near-optimal allocation of the resources based on this relation. The power allocation problem in NOMA systems can be modeled as a variation of a KP, and this can yield a near-optimal solution based on such a greedy heuristic.

In this paper, we concentrate on the problem's stochastic nature and propose an adaptive RL-based solution. More specifically, by invoking a technique within the OMA paradigm, we see that partitioning problems can be solved even in highly stochastic environments. They thus constitute valuable methods for handling the behavior of components in a NOMA system. In particular, we shall show that such methods are compelling in resolving the task of grouping the users. Indeed, even though the number of possible groupings can be exponentially large, the OMA-based scheme yields a remarkably accurate result within *a few hundred iterations*. This constitutes the first phase of our solution. It is pertinent to mention that the solution is unique, and *that we are not aware of any analogous RL-based solution for this phase of NOMA systems*.

The second phase groups users with different channel behaviors, and allocates power to the respective users. Here, we observe that the power allocation problem can be mapped onto a variation of a KP. Although the types of reported KPs are numerous, our specific problem is more analogous to a *linear* KP. By observing this, we are able to resolve the power allocation by solving a linear (in the number of users) number of algebraic equations, all of which are also *algebraically linear*. This two-step solution constitutes a straightforward, but comprehensive strategy. Neither of them, individually or together, has been considered in the prior literature.

The paper is organized as follows. In Section 2, we depict the configuration of the adopted system. Then, in Section 3, we formulate and analyze the optimization problem. Section 4 details the proposed solution for the optimization problem. We briefly present numerical results in Section 5, and conclude the paper in Section 6.

2 System description

Consider a simplified single-carrier down-link cellular system that consists of one base station (BS) and K users that are to be divided into N groups for NOMA operation. NOMA is applied to each group, but different groups are assigned to orthogonal resources. For example, one BS assigns a single frequency band to the K users. The users are to be grouped in N groups, each of which occupies a time slot. User k is denoted by U_k where $k \in \mathcal{K} = \{1,2,\ldots,K\}$. Similarly, the set of groups are denoted by $\mathcal{G} = \{g_n\}$, $n \in \mathcal{N} = \{1,2,\ldots,N\}$, where g_n is the set of users inside the n-th group. The groups are mutually exclusive and collectively exhaustive, and thus, $g_n \cap g_o = \emptyset$ with $n \neq o$. When a User U_k belongs to Group n, we use the notation $U_{n,k}$ to refer to this user and its group. We adopt the simplified notation U_k to refer to a user when the user's group is trivial or undetermined. Thus, if we have 4 users in the system, User 1 and User 3 could belong to Group 1, and User 2 and User 4 belong to group 2. In this case, when we want to refer to User 1 without its group, we use U_1. Likewise, when we want to mention User 4 belonging to Group 2, we apply $U_{2,4}$. For mobility, the users are expected to move within a defined area. The user behavior in a university or an office building are examples of where the user behavior coincides with our mobility model.

2.1 Channel Model

The channel model coefficient for U_k is denoted by $h_k(t)$ and refers to the channel fading between the BS and U_k along time. The channel coefficient is generated based on the well-recognized mobile channel model, which statistically follows a Rayleigh distribution [8]. The parameters of the channel configuration will be detailed in the section describing the numerical results. Note that the LA solution to be proposed can handle a non-stationary stochastic process, and the solution proposed in this work is distribution-independent. Therefore, the current Rayleigh distribution can be replaced by any other channel model, based on the application scenario and environment.

2.2 Signal Model

Based on the NOMA concept, the BS sends different messages to the users of a group in a single time slot via the same frequency band. Consequently, the received signal y_k at time t for $U_{n,k}$ can be expressed as

$$y_k(t) = \sqrt{p_{n,k}} h_k(t) s_k + \sum_{e=1}^{|g_n|-1} \sqrt{p_{n,e}} h_k(t) s_e + n_k, \tag{1}$$

where e is the index of the users in the set $g_n \backslash U_{n,k}$, which is the complementary set of $U_{n,k}$ in g_n. $|g_n|$ returns the number of users in g_n. The received signal $y_k(t)$ has three parts, including the signal intended for $U_{n,k}$, the signal from all users other than $U_{n,k}$ in the same group, and the additive white Gaussian noise (AWGN) $n_k \sim \mathcal{CN}(0,\sigma_k^2)$ [10]. The transmitted signal intended for $U_{n,k}$ and $U_{n,e}$ is given by s_k and $s_e \sim \mathcal{CN}(0,1)$ respectively. $p_{n,k}$ is the allocated power for $U_{n,k}$, and the total power budget for group g_n is given by P_n.

The BS' signals are decoded at the users through SIC by means of channel coefficients in an ascending order [9]. As a result, through SIC, a user with a good channel quality can remove the interference from the users of poor channel quality, while users of poor channel quality decode their signals without applying SIC. Hence, for the User $U_{n,k}$, successful SIC is applied when $|h_{n,w}(t)|^2 \leq |h_{n,k}(t)|^2$ fulfills, where w is the index of the users that have lower channel coefficients than User k in the user Group g_n.

3 Problem Formulation

In this section, we formulate the problem to be solved. The problem is divided into two sub-problems. Specifically, in the first problem, we cluster the users into categories based on the time average of the channel coefficients. In the second step, we group the users based on the learned categories and solve the resultant power allocation problem.

3.1 Problem Formulation for the Clustering Phase

To initiate discussions, we emphasize that the channel coefficients of the users in a group need to be as different as possible so as to achieve successful NOMA operation. To group the users with different coefficients, we thus first cluster the users with *similar* coefficients, and then select one user from each cluster to formulate the groups. The first problem, the clustering problem, is formulated in this subsection. The problem for user grouping, together with power allocation, is formulated later.

The criterion that we have used for clustering the users is the *time average* of the channel coefficients, $\overline{h_k(t)}$. The reason motivating this is because the user grouping is computationally relatively costly, and the fact that the environment may change rapidly, i.e., $h(t)$ might change after channel sounding. If we cluster the users according to the time average, we can reduce the computational cost, and at the same time, capture the advantages of employing NOMA statistically.

We consider clustering users to clusters of the same size, where the number of the clusters is $L_c = K/N$, and where L_c and N are integers[4]. Let q_c be the set of users in Cluster c, where $c \in [1,2,\ldots,L_c]$ is the index of the cluster. Clearly, for the clustering problem, the difference of coefficients in each cluster needs to be minimized, and the problem can be formulated as

$$\min_{\{\varphi_{c,k}\}} \sum_{c=1}^{L_c} \sum_{k=1}^{K} \varphi_{c,k} |\overline{h_k(t)} - E_c|, \tag{2a}$$

$$\text{s.t.} \sum_{c=1}^{L_c} \sum_{k=1}^{K} \varphi_{c,k} = K, \;\; c \in \mathcal{C}, k \in K, \tag{2b}$$

$$\sum_{k=1}^{K} \varphi_{c,k} = N, \;\; \forall c, \tag{2c}$$

[4] In reality, if L_c is not an integer, we can add dummy users to the system so as to satisfy this constraint. *Dummy users* are virtual users that are not part of the real network scenario, but are needed for constituting equal-sized partitions in the clustering phase.

where $\varphi_{c,k}$ is an indicator function showing the relationship of users and clusters, as $\varphi_{c,k} = 1$ when U_k belongs to Cluster c, and 0 otherwise. Additionally, the mean value of the channel fading in each cluster is denoted by the parameters E_c and δ, which are given by $E_c = \frac{1}{\delta}\sum_{k=1}^{K} \overline{h_k(t)}\varphi_{c,k}$, and $\delta = \sum_{k=1}^{K} \varphi_{c,k}$ respectively. To explain the above equation, we mention that Eq. (2a) states the objective function. Specifically, for all the clusters, we want to minimize the difference of channel coefficients between the users within each cluster. Eq. (2b) and Eq. (2c), state a description of the variable $\varphi_{c,k}$ for User k in c. Hence, the sum of the variables $\varphi_{c,k}$ needs to be equal to the number of users, meaning that all the users need to be a part of a single cluster, and in each cluster, there needs to be an equal number of users.

The result of the clustering problem, i.e., the $\{\varphi_{c,k}\}$ that minimizes the objective function, formulates L_c sets of users, each of which has exactly N users.

3.2 Problem Formulation for the Power Allocation

From the output of the clustering, we know which users that are similar. Thus, when we take one user from each cluster and construct N groups, the size of each group is L_c. Without loss of generality, we can assume that the average channel coefficients are sorted in ascending order, i.e., $\overline{h_1(t)} \leq \overline{h_2(t)} \leq \ldots \leq \overline{h_K(t)}$. If we consider user grouping and power allocation based on average channel coefficients, the reduces to:

$$\max_{\{g_n\},\{p_{n,k}\}} \quad R = \sum_{k=1}^{K} b\log_2\left(1 + \frac{p_{n,k}|\overline{h_{n,k}}|^2}{I_{n,k}+\sigma^2}\right) \tag{3a}$$

$$\text{s.t.} \quad g_n \cap g_o = \emptyset, \;\; n \neq o, \;\; n,o \in \mathcal{N}, \tag{3b}$$

$$\sum_{j,\forall U_j \in g_n} p_{n,j} \leq P_n, \;\; n \in \mathcal{N}, \tag{3c}$$

$$R_{n,j}(t) \geq R_{QoS}, \;\; j \in \mathcal{K}, n \in \mathcal{N}, \tag{3d}$$

$$\overline{h_i(t)} > \overline{h_j(t)}, \;\; \forall i > j, \;\; i,j \in \mathcal{K}, \tag{3e}$$

$$|g_n \cap q_c| = 1, \;\; \forall c, \; \forall n, \tag{3f}$$

$$\sum_{j,\forall U_j \in g_n} \tau_{n,j} = L_c, \;\; \forall n, \tag{3g}$$

$$\sum_{n,\forall n \in \mathcal{N}} \sum_{j,\forall U_j \in g_n} \tau_{n,j} = NL_c. \tag{3h}$$

In Eq. (3a), $I_{n,k} = \sum_{\substack{j, \\ \forall j>k, \; \{U_j,\, U_k\} \in g_n}} |\overline{h_k}|^2 p_{n,j}$ is the interference to User k in Group n. In Eq. (3b), we state that the groups need to be disjoint. Hence, one user can only be in one group. In (3c), we address the constraint for the power budget. The QoS constraint is given in Eq.(3d), where $R_{n,j}(t)$ is the achievable data rate for User j in Group n, and Eq. (3e) gives the SIC constraint. The constraint in Eq. (3f) specifies that only a single user is selected to formulate a group from each cluster. In Eq. (3g), we introduce an indicator $\tau_{n,k}$, stating whether U_k is in Group n, as $\tau_{n,k} = 1$ when U_k belongs to Group n, and 0 otherwise. Furthermore, all users should belong to a certain group, which is given in Eq. (3h). Table 1 summarizes the notation.

Notation	Description	Notation	Description
$h, h(t)$	Channel coefficient, and h for t	$p_{n,k}$	Allocated power for $U_{n,k}$
$h_k, h_{n,k}$	h for U_k and $U_{n,k}$	n_k, σ^2	AWGN at U_k and Gaussian noise
$\overline{h_k}(t), \overline{h_{n,k}}(t)$	The mean of h for U_k	$h_k(t), h_{n,k}(t)$	h for U_k and $U_{n,k}$ at t
K	Total number of users	$I_{n,k}$	Interference from other users to $U_{n,k}$
N	Total number of groups	E_c	Mean of channel fading in q_c
$\mathcal{K}$	Set of user indexes	R	Total data rate (capacity)
$\mathcal{N}$	Set of group indexes	S	Number of states per action
$\mathcal{G}$	Set of groups	R_{QoS}	Minimum required data rate for a user
g_n	Set of users inside the n-th group	v_U, v_L	Mobility factor and speed of light
$\lvert g_n \rvert$	Number of users inside g_n	f_c, f_d	Carrier and Doppler frequency
$U_k, U_{n,k}$	User k and user k in group n	L_c	Number of clusters
$g_n \backslash U_{n,k}$	The complementary set of users in set g_n	q_c	Set of users in cluster c
$\emptyset$	Empty set	$\mathcal{C}$	Set of clusters
$y_k, y_k(t)$	Signal from BS at U_k and U_k at time t	$\varphi_{c,k}$	Indicator of whether U_k is in cluster c
s_k	Transmitted signal intended for U_k	δ	Number of $\varphi_{c,k} = 1$ for a cluster
P_n	Power budget for g_n	b	Channel bandwidth
$\tau_{n,k}$	Indicator of whether U_k is in group n	r_k	Rank of U_k
Δ_t	Time period for average of h	Υ_k	Ranking category of user k
ε_k	Index of the current state of user k	Θ_k	Cluster of U_k
$Q = (U_a, U_b)$	Input *query* of users to the EOMA	W, $Mbps$	Watt and Megabits per second
r	Rank	c	Index of the set of clusters

Table 1. Summary of notations

4 Solution to User Grouping and Power Allocation

The problem of grouping and power allocation in NOMA systems is two-pronged. Therefore, in Section 4.1, we only consider the first issue of the two, namely the grouping of users. We will show that our solution can handle the stochastic nature of the channel coefficients of the users, while also being able to follow changes in their channel behaviors over time. This will ensure that the system will be able to follow the nature of the channels in a manner that is similar to what we will expect in a real system. Thereafter, in Section 4.2, we will present our solution to the power allocation problem. Once the groups have been established in Section 4.1, we can utilize these groups to allocate power among them either instantaneously, or over a time interval using a greedy solution to the problem.

4.1 Clustering Through EOMA

The family of OMA algorithms are based on *tabula rasa* Reinforcement Learning. Without any prior knowledge of the system parameters, the channels, or the clusters, (as in our case), the OMA self-learns by observing, over time, the Environment that it interacts with. For our problem, the communication system constitutes the Environment, which can be observed by the OMA through, e.g., channel sounding. By gaining knowledge from the system behavior and incrementally improving through each interaction with the Environment, the OMA algorithms are compelling mechanisms for solving complex and stochastic problems. In the OMA, the users of our system need to be represented as abstract objects. Therefore, as far as the OMA is concerned, the users are called “objects”. The OMA algorithms require a number of states per action, indicated by S. For the LA, an action is a solution that the algorithm can converge to. In our system, the actions are the different clusters that the objects may belong to. Hence,

based on the current state of an object, we know that object's action, which is equal to its current cluster in our system. Therefore, each object, or user in our case, has a given state indicated by $\varepsilon_k = \{1, 2, ..., SL_c\}$, where ε_k denotes the current state of U_k, S is the number of states per action, and L_c is the number of clusters. Clearly, because we have L_c clusters, the total number of possible states is SL_c. To indicate the set of users inside Cluster c, where $c \in [1, 2, \ldots, L_c]$, we have q_c. The cluster for a given User, k, is represented by Θ_k, where the set of clusters is denoted by $\mathcal{C}$ and $\Theta_k \in \mathcal{C} = \{q_1, q_2, \ldots, q_{L_c}\}$.

Algorithm 1 Clustering of Users

Require: $h_k(t)$ for all users K
  while not converged **do** // Converged if all users are in the two innermost states of any action
    for all K **do**
      Rank the users from 1 to K // 1 is given to the user with lowest h (K to the highest)
    end for
    for $\frac{K}{N}$ pairs (U_a, U_b) of K **do** // The pairs are chosen uniformly from all possible pairs
      if $\Upsilon_a = \Upsilon_b$ **then** // If U_a and U_b have the same ranking category
        if $\Theta_a = \Theta_b$ **then** // If U_a and U_b are clustered together in the EOMA
          Process Reward
        else // If U_a and U_b are not clustered together in the EOMA
          Process Penalty
        end if
      end if
    end for
  end while // Convergence has been reached

The states are central to the OMA algorithms, and the objects are moved in and out of states as they are penalized or rewarded in the Reinforcement Learning process. When all objects have reached the two innermost states of an action, we say that the algorithm has converged. When convergence is attained, we consider the solution that the EOMA algorithm has found to be sufficiently accurate. In the EOMA, the numbering of the states follows a certain pattern. By way of example, consider a case of three possible clusters: the first cluster of the EOMA has the states numbered from 1 to S, where the innermost state is 1, the second innermost state is 2, and the boundary state is S. The second cluster has the innermost state $S+1$ and the second innermost state $S+2$, while the boundary state is $2S$. Likewise, for the third cluster, the numbering will be $2S+1$ for the innermost and $2S+2$ for the second innermost state, while $3S$ is the boundary state.

Algorithm 1 presents the overall operation for the clustering of the users. The functionality for reward and penalty, as the EOMA interacts with the NOMA system, are given in Algorithms 2 and 3 respectively. In the algorithms, we consider the operation in relation to a pair of users U_a and U_b, and so $Q = \{(U_a, U_b)\}$. The EOMA considers users in pairs (called *queries*, denoted by Q). Through the information contained in their pairwise ranking, we obtain a clustering of the users into the different channel categories. For each time instant, Δ_t, the BS obtains values of $h_k(t)$ through channel sounding, and we use the average of Δ_t samples as the input to the EOMA ($\overline{h_k(t)}$). The

BS then ranks the users, indicated by $r_k = \{1, 2, ..., K\}$, where each U_k is given a single value of r_k for each Δt. For the ranks, $r_k = 1$ is given to the user that has the lowest channel coefficient compared to the total number of users, and $r_k = K$ is given to the user with the highest channel coefficient of the users. The others are filled in between them with ranks from worst to best. Furthermore, the values of these ranks corresponds to ranking categories, denoted by Υ_k for U_k, where $\Upsilon_k = \{r \in [1,N] = 1, r \in [1+N, 2N] = 2, r \in [1+2N, 3N] = 3, \ldots, r \in [K-N+1, K] = L_c\}$. In this way, even if the users have similar channel conditions, they will be compared, and the solution can work on finding the current best categorization of the K users for the given communication scenario. As depicted in Algorithm 1, we check the users' ranking categories in a pairwise manner. If the users in a pair (query) are in the same ranking category, they will be sent as a query to the EOMA algorithm. The EOMA algorithm will then work on putting the users that are queried together in the same cluster, which, in the end, will yield clusters of users with similar channel coefficients. More specifically, if two users have the same ranking category, they are sent as a query to the EOMA and the LA is rewarded if these two users are clustered together (penalized if they are not together).

Algorithm 2 Process Reward

Require: $Q = (U_a, U_b)$ // A query (Q), consisting of U_a and U_b
Require: The state of U_a (ε_a) and U_b (ε_b)
  if $\varepsilon_a \; mod \; S \neq 1$ **then** // U_a not in innermost state
    $\varepsilon_a = \varepsilon_a - 1$ // Move U_a towards innermost state
  end if
  if $\varepsilon_b \; mod \; S \neq 1$ **then** // U_b not in innermost state
    $\varepsilon_b = \varepsilon_b - 1$ // Move U_b towards innermost state
  end if
  return The next states of U_a and U_b

When the algorithm converges to obtain the groups that are needed for the power allocation, we rank the users within each cluster based on $\overline{h_k(t)}$ that was obtained in the clustering process, and then formulate the groups that consist of one user from each cluster with the same rank.

4.2 Power Allocation Through a Greedy Solution

Once the grouping of the users has been established, we can allocate power to different users in such a way that the joint data rate (R) is maximized. There are numerous ways of power allocation in various communication scenarios [5, 10]. The power allocation can be replaced by any other algorithm and will not change the nature of the Reinforcement Learning procedure. However, in this paper, we will consider the problem of power allocation as a variation of the KP, and solve it through a greedy solution.

Our aim for the greedy solution is that of maximizing the total data rate of the system. Thus, the weakest user will always be limited to the minimum required data

Algorithm 3 Process Penalty

Require: $Q = (U_a, U_b)$ // A query (Q), consisting of U_a and U_b
Require: The state of U_a (ε_a) and U_b (ε_b)
  if $\varepsilon_a \ mod \ S \neq 0$ and $\varepsilon_b \ mod \ S \neq 0$ **then** // Neither of the users are in boundary states
    $\varepsilon_a = \varepsilon_a + 1$, $\varepsilon_b = \varepsilon_b + 1$ // Move U_a and U_b towards boundary state
  else if $\varepsilon_a \ mod \ S \neq 0$ and $\varepsilon_b \ mod \ S = 0$ **then** // U_b in boundary state but not U_a
    $\varepsilon_a = \varepsilon_a + 1$, $temp = \varepsilon_b$
    $x =$ unaccessed user in cluster of U_a which is closest to boundary state
    $\varepsilon_x = temp$, $\varepsilon_b = \varepsilon_a$
  else if $\varepsilon_b \ mod \ S \neq 0$ and $\varepsilon_a \ mod \ S = 0$ **then** // U_a in boundary state but not U_b
    $\varepsilon_b = \varepsilon_b + 1$, $temp = \varepsilon_a$
    $x =$ unaccessed user in cluster of U_b closest to boundary state
    $\varepsilon_x = temp$, $\varepsilon_a = \varepsilon_b$
  else // Both users are in boundary states
    $\varepsilon_y = \varepsilon_{\{a \ or \ b\}}$ // y equals a or b with equal probability, and y is the staying user
    $\varepsilon_z = \varepsilon_{\{a \ or \ b\}}$ // z is the moving user, and is a if b was chosen as y (b if a was chosen)
    $temp = \varepsilon_z$
    $x =$ unaccessed user in cluster of U_y closest to boundary state
    $\varepsilon_x = temp$
    $\varepsilon_z = \varepsilon_y$ // Move U_z to cluster of U_y
  end if
  return The next states of U_a and U_b

rate. The heuristic involves allocating the majority of the power to the users with higher values of h, and this will result in a higher sum rate for the system. Consequently, the stronger users are benefited more from the greedy solution than those with weaker channel coefficients. However, the weak users' required data rate is ensured and can be adjusted to the given scenario. The formal algorithms are not explicitly given here in the interest of brevity, and due to space limitations. They are included in [7].

5 Numerical Results

The techniques explained above have been extensively tested for numerous numbers of users, power settings etc., and we give here the results of the experiments. In the interest of brevity, and due to space limitations, the results presented are a very brief summary of the results that we have obtained. More detailed results are included in [7] and in the Doctoral Thesis of the First Author.

We employed Matlab for simulating the values of the channel coefficient, h. Additionally, we invoked a Python script for simulating the LA solution to the user grouping and the greedy solution to power allocation. The numerical results for the power allocation solution are based on the results obtained from the EOMA clustering and grouping. For the simulations, we used a carrier frequency of $5.7\,GHz$ and an underlying Rayleigh distribution for the corresponding values of $h(t)$. For the mobility in our model, we utilized a moving pace corresponding to the movement inside an office building, i.e., $v_U = 2\ km/h$. We sampled the values of h according to $\frac{1}{2f_d}$, where f_d is the Doppler frequency and f_c is the carrier frequency. The Doppler frequency can be expressed as

$f_d = f_c(\frac{v_U}{v_L})$ and v_L is the speed of light. Therefore, in the following figures, we use "Sample Number" as the notation on the X-axis. Fig. 1 illustrates the snap-shot of h values for four users and the principle for the simulation when the number of users increased.

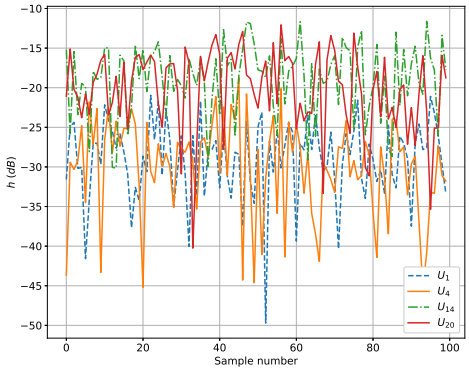


Fig. 1. Example of the simulated $h(t)$ for four different users. In the interest of clarity, and to avoid confusion, we did not plot all the 20 users.

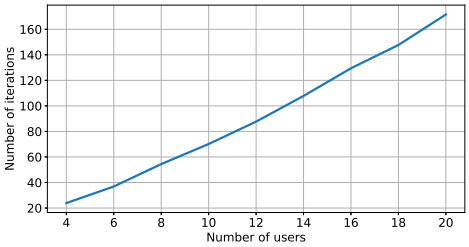


Fig. 2. A plot of the average number of iterations needed before convergence, as a function of number of users, where there were two users in each group. This was obtained by executing 100 independent experiments.

For evaluating the simulation for the clustering phase, we recorded whether or not the LA were able to determine the clusters that corresponded to the minimized difference between the users in a cluster, based on the users' mean values of h in the simulations. Remarkably, in the simulation, the EOMA yielded a 100 % accuracy in which the learned clustering was identical to the unknown underlying clustering in every sin-

gle run for the example provided with $-10dB$ difference between values of h within the different clusters. This occurred for groups of sizes $4, 6, 8, 10, 12, 14, 16, 18$ and 20, where the number of users in a group was equal to two. The difference between the users can be replaced by any "equivalent metric", and it should be mentioned that these values were only generated for testing the solution, since in a real scenario, the "True partitioning" is always unknown. The number of iterations that it took the EOMA to achieve 100 % accuracy for the different number of users is depicted in Fig. 2. Notably, the EOMA retains its extremely high accuracy as the number of users increased, and yielded 100 % accuracy both for 4 users as well as 20 users.

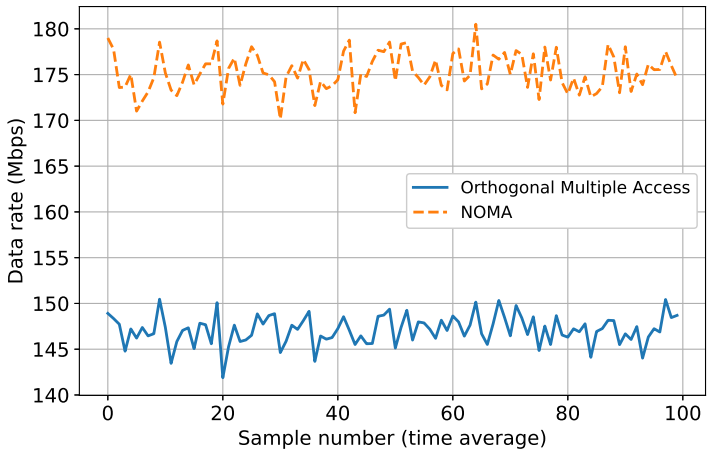


Fig. 3. Data rate for orthogonal multiple access compared to NOMA for time averages of h. Based on averages over 500 samples of h.

The simulation for the greedy solution to the power allocation phase, was carried out based on the groups established in the LA solution. Again, in the interest of brevity, we only report the results for the cases with 20 users in total and 2 users in each group, and more extensive results are included in [7] and in the Doctoral Thesis of the First Author. The optimal power allocation for 2 users in a group was obtained when we gave the minimum required power to the user with smaller h value and then allocated the rest to the user with lager h value. The optimality of the greedy algorithm for the 2 user-group case was verified by an alternate independent exhaustive search. For illustrating the advantages of our NOMA greedy solution when NOMA was employed, we compared it with the data rate that would be achieved with orthogonal multiple access[5].

[5] The achieved data rate for User k in Group n in orthogonal multiple access is given by $R_{n,k} = \frac{1}{2}\log_2\left(1+\frac{P_n|h_{n,k}|^2}{\sigma^2}\right)$. The factor $\frac{1}{2}$ is due to the multiplexing loss when 2 users share the orthogonal resource.

In Fig. 3, we depict the results obtained for the greedy NOMA solution together with the orthogonal multiple access, for an average over $\Delta t = 5$ samples of h. Further, with regard to the parameters used, the data rate for the simulations depicted in Fig. 3 was based on the following configuration: The minimum required data rate was configured to 2.0 $Mbps$, the noise to 10^{-8} W, the bandwidth to 1 MHz, and the power level for all groups to 0.125 W. As illustrated, the simulation results obtained show that the greedy solution to the power allocation is higher than the data rate achieved with orthogonal multiple access. From the graph in Fig. 3 we see that the average difference between the orthogonal multiple access and NOMA was approximately 28.17 $Mbps$.

6 Conclusions

In this paper, we have proposed a novel solution to the user grouping and power allocation problems in NOMA systems, where we have considered the stochastic nature of the users' channel coefficients. The grouping has been achieved by using the *tabula rasa* Reinforcement Learning technique of the EOMA, and the simulation results presented demonstrate that a 100 % accuracy for finding clusters of similar $h(t)$ over time can be obtained within a limited number of iterations. With respect to power allocation, we proposed a greedy solution, and again the simulation results confirm the advantages of the NOMA solution. Our solutions offer flexibility, as both the grouping and the power allocation phases, can be used as stand-alone components of a NOMA system.

References

1. Cui, J., Ding, Z., Fan, P., Al-Dhahir, N.: Unsupervised Machine Learning-Based User Clustering in Millimeter-Wave-NOMA Systems. IEEE Transactions on Wireless Communications **17**(11), 7425–7440 (Nov 2018)
2. Gale, W., Das, S., Yu, C.T.: Improvements to an Algorithm for Equipartitioning. IEEE Transactions on Computers **39**(5), 706–710 (May 1990)
3. Glimsdal, S., Granmo, O.: 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)
4. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack problems. Springer, Berlin (2004)
5. Liu, Y., Elkashlan, M., Ding, Z., Karagiannidis, G.K.: Fairness of User Clustering in MIMO Non-Orthogonal Multiple Access Systems. IEEE Communications Letters **20**(7), 1465–1468 (July 2016)
6. Liu, Y., Qin, Z., Elkashlan, M., Ding, Z., Nallanathan, A., Hanzo, L.: Nonorthogonal Multiple Access for 5g and Beyond. Proceedings of the IEEE **105**(12), 2347–2381 (Dec 2017)
7. Omslandseter, R. O., Jiao, L., Liu, Y. Oommen, B. J.: An Efficient and Fast Reinforcement Learning-Based Solution to the Problem of User Grouping and Power Allocation in NOMA Systems. Unabridged version of this paper. To be submitted for publication.
8. Pätzold, M.: Mobile Radio Channels. Wiley, Chichester, 2nd ed. edn. (2012)
9. Pischella, M., Le Ruyet, D.: Noma-Relevant Clustering and Resource Allocation for Proportional Fair Uplink Communications. IEEE Wireless Communications Letters **8**(3), 873–876 (June 2019)
10. Xing, H., Liu, Y., Nallanathan, A., Ding, Z., Poor, H.V.: Optimal Throughput Fairness Tradeoffs for Downlink Non-Orthogonal Multiple Access Over Fading Channels. IEEE Transactions on Wireless Communications **17**(6), 3556–3571 (June 2018)

D.2 User Grouping and Power Allocation in NOMA Systems: A Novel Semi-supervised Reinforcement Learning-based Solution

This paper has been published as:

R. O. Omslandseter, L. Jiao, Y. Liu, and J. B. Oommen, "User Grouping and Power Allocation in NOMA Systems: A Novel Semi-Supervised Reinforcement Learning-Based Solution," *Pattern Analysis and Applications*, vol 26, pp.1–17, Springer London, July 2022.
DOI: https://doi.org/10.1007/s10044-022-01091-2

Noname manuscript No.
(will be inserted by the editor)

User Grouping and Power Allocation in NOMA Systems: A Novel Semi-Supervised Reinforcement Learning-based Solution

Rebekka Olsson Omslandseter ·
Lei Jiao ·
Yuanwei Liu ·
B. John Oommen

Received: date / Accepted: date

Abstract In this paper, we present a pioneering solution to the problem of user grouping and power allocation in Non-Orthogonal Multiple Access (NOMA) systems. The problem is highly pertinent because NOMA is a well-recognized technique for future mobile radio systems. The salient and difficult issues associated with NOMA systems involve the task of grouping users together into the pre-specified time slots, which are augmented with the question of determining how much power should be allocated to the respective users. This problem is, in and of itself, NP-hard. Our solution is the first reported Reinforcement Learning (RL)-based solution, which attempts to resolve parts of this issue. In particular, we invoke the Object Migration Automaton (OMA) and one of its variants to resolve the grouping in NOMA systems. Furthermore, unlike the solutions reported in the literature, we do not assume prior knowledge of the channels' distributions, nor of their coefficients, to achieve the grouping/partitioning. Thereafter, we use the consequent groupings to heuristically infer the power allocation. The simulation results that we have obtained confirm that our learning scheme can follow the dynamics of the channel coefficients efficiently, and that the solution is able to resolve the issue dynamically.

R. O. Omslandseter and L. Jiao
Address of the first two authors: Department of Information and Communication Technology, University of Agder, Jon Lilletuns vei, 4879 Grimstad, Norway. E-mail: rebekka.o.omslandseter@uia.no, lei.jiao@uia.no.

Yuanwei Liu
Address: School of Electronic Engineering and Computer Science, Queen Mary University of London, Mile End Road, London, E1 4NS, U.K. E-mail: yuanwei.liu@qmul.ac.uk.

B. John Oommen
Chancellor's Professor; *Life Fellow: IEEE* and *Fellow: IAPR*. Address: School of Computer Science, Carleton University, Ottawa, Canada: K1S 5B6. This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway. E-mail: oommen@scs.carleton.ca.

Keywords Learning Automata · Non-Orthogonal Multiple Access · Object Migration Automaton · Object Partitioning · Semi-Supervised Reinforcement Learning

1 Introduction

The Non-orthogonal Multiple Access (NOMA) paradigm has been established as a promising technique to meet the future requirements of wireless capacity [1]. As user demands are increasing due to the ever-increasing range of applications and technologies, (such as the Internet of things) (IoT), NOMA constitutes a valuable solution, as more users can be multiplexed together in the same orthogonal Resource Block (RB) [2]. With NOMA, the diversity of the user's channel and power are exploited through Successive Interference Cancellation (SIC) techniques in receivers [3]. The RB sharing introduces questions as to which users are the most ideal candidates to be grouped together, so as to obtain the maximum gain of capacity. Additionally, the power level of the signal intended for each user is a crucial component for successful SIC in NOMA operation. Therefore, the performance of NOMA is highly dependent on both the user grouping *and* the subsequent power allocation.

The user grouping and power allocation problems in NOMA systems are, in general, inter-twined and intricate. The user grouping problem, in and of itself, introduces a combinational in-feasibility when the number of users increases. In addition, users in NOMA systems can suffer from both inter- and intra-channel interference, constituting the non-convex property of power allocation in NOMA systems [3]. Furthermore, the channel conditions and user behavior in communication scenarios have a random nature, complicating the problem. Consequently, the foundation for grouping, and thus for power allocation, can change rapidly. Therefore, in addition to searching for instantaneous optimization, it is necessary for a modern communication system to accommodate and adapt to such changes.

In recent years, the fields of Machine Learning (ML), including Reinforcement Learning (RL), have been exploding, which provides new opportunities for facilitating communication systems with more capacity in terms of automation. With some insight, one sees that the problem of user grouping in NOMA systems, is similar in nature and with regard to the solution space, to a classic problem, namely, to the Object Partitioning Problem (OPP). The OPP concerns grouping objects into sub-collections and attempts to optimize a related objective function to obtain a near-optimal grouping [4]. When it concerns RL-based solutions for the OPP, many recent studies have been carried out for Equi-Partitioning Problems (EPPs). EPPs are a subset of OPPs, where all the groups (referred to as partitions) need to be equi-sized.

Among the ML solutions, Enhanced Object Migration Automata (EOMA)[1] is a technique that performs well for solving different variants of EPPs [5].

[1] Object Migration Automata and its abbreviation, OMA, should not be confused with Orthogonal Multiple Access, often also abbreviated OMA in the Literature. In this paper, the abbreviation OMA (and its variants) refers to the ML algorithm.

Further advancements to the EOMA algorithm, that can be found in [6] and [7], are the Pursuit Enhanced Object Migration Automata (PEOMA) and the Transitivity Pursuit Enhanced Object Migration Automata (TPEOMA), respectively. These OMA-based algorithms can partition a set of users into disjoint groups through the use of Learning Automata (LA)[2] instances, which can handle stochastic behavior. LA instances learn through the concept of RL in a semi-supervised manner. These are powerful techniques applicable in highly dynamic environments, dealing with problem instances that are akin to the underlying ones encountered by users in NOMA systems.

An LA is a decision-making algorithm that can sequentially learn the optimal action from a set of actions in a stochastic environment. At each time instant, an action is chosen by the LA, and serves as the input to the environment. The environment then responds to the action chosen by the LA, by a feedback that is usually a reward or a penalty to the action. Based on both the response of the environment and the current state of the LA, the LA adjusts its action selection strategy for future interactions. Initial LA was designed in [21] with a fixed structure, where the state update and the decision functions were time-invariant. Later, Variable Structure Stochastic Automata (VSSA) were developed, such as the linear reward-penalty scheme, the linear reward-inaction scheme, the linear inaction-penalty scheme, and the linear reward-penalty scheme [22], [23]. Schemes that apply nonlinear functions have also been designed and analyzed [22], [23], [24], where the updating functions can either be of continuous or discretized [25], [26]. In addition, the Markovian representation of the states in LA can be either absorbing or ergodic [22], [27], where the latter adapts better to non-stationary environments where the reward probabilities are time-variant. The state-of-the-art of the field of LA, is reported in [28], [29].

The task of allocating power to the different users of a group in NOMA systems further complicates the NOMA operations. Depending on the objective of power allocation, the formulation and complexity can be quite different. Various solutions can be adopted for distinct problems, and heuristics-based solutions can be quite pertinent to such a highly complex problem.

1.1 Motivation of this Paper

In communications, a particular distribution for the channel fading is often assumed, e.g., Rayleigh fading, which involves a stochastic process in which we observe the channel, as time proceeds. When the channel coefficient, h, is assumed to follow a specified distribution that is time-invariant, it is equivalent to assuming that the stochastic process follows a random and stationary process. In previous solutions, although channel fading was assumed to be following a certain random distribution, user grouping and power allocation

[2] The abbreviation LA is used interchangeably throughout the paper, referring to both the field of Learning Automata and the Learning Automaton itself, depending on the context in which it appears.

were traditionally carried out based on an instantaneous sample from the distribution, and thus a constant, h, was assumed to have been known, and was utilized for optimization. In other words, the stochastic behavior of channels was not handled in the prior grouping and power allocation.

Although channel sounding is not an expensive operation, it may not be carried out frequently enough to follow the instantaneous changes of the channels in certain systems. Therefore, basing a system on the most recent channel sounding result for optimization, may not be a statistically-prudent strategy. Furthermore, due to the complexity of optimization problems, in practice, the system may not prefer to solve the optimization problem frequently based on, e.g., instantaneous channel sounding results every time when they are available. In addition, the overall statistics of the channel may even change over time due to, for example, mobility. Therefore, we need a more reasonable base for the channel coefficients for optimization, and at the same time, require a more computationally-effective and adaptive solution.

1.2 Contributions of this Paper

In this paper, we study the problem considering the issue's stochastic nature, and propose an adaptive solution based on RL. To be specific, by incorporating a technique from within the OMA paradigm, partitioning problems can be solved even in environments with a highly stochastic nature. Hence, OMA algorithms constitute valuable methods for handling the behavior of components in a NOMA system. In particular, we shall show that such methods are powerful in resolving the task of grouping the users. Indeed, even though the number of possible groupings can be exponentially large, the OMA schemes can yields a remarkably accurate result, which can be achieved within a few hundred iterations! This constitutes the first phase of our solution.

The second phase groups users with different channel behaviors and allocates power to the respective users. For power allocation, we adopt the objective of maximizing the sum rate, and propose two heuristic-based algorithms. Other objective functions and the corresponding solutions may also be employed for power allocation, depending on the system's demand. This two-step solution constitutes a straightforward but comprehensive strategy, which has not been considered in the prior literature.

Our proposed solution handles random stationary environments and can learn from the environment and adjust user groupings adaptively, based on the time-averaged values of the channel's coefficients. Instead of grouping users and allocating power to users based on instantaneous measurements, which is not practical due to the complexity and stochastic nature of the problem domain, the proposed solution employs user grouping according to the time average of the communication environment. Further, based on the obtained groups, heuristic-based schemes resolve power allocation in NOMA systems. In addition, when the statistics of the environment changes, the RL algorithm can follow them so that the groupings of the users can be updated. The beauty

of the proposed algorithm is that it requires no prior knowledge of the channel, *and* that the learning and adaptation are carried out while the communication system is in operation.

The reader will also observe that the problem is two-pronged. Firstly, it involves the grouping of the users, and thereafter, secondly, the corresponding power allocations. With respect to the first prong, our solution converges arbitrarily close to the optimal clustering. This, of course, does not address the power allocation problem. To address the second prong, we have resorted to straightforward heuristic-based algorithms.

Our contributions can thus be summarized as follows:

1. We study the user grouping and power allocation problem in stochastic environments. This real-life scenario is hardly addressed in the literature.
2. Through a two-step solution, the user grouping and power allocation problems are solved through a RL technique and heuristic solutions, respectively. The solution is adaptive to changes in the environment. Additionally, without prior knowledge of the channel, the system can learn the knowledge when the system is in operation, and thus both these solutions, can be implemented on the fly.
3. We provide fairly extensive simulation results to illustrate the effectiveness and the strength of RL for problems in NOMA systems.

1.3 Organization of the Paper

The paper is organized as follows[3]. First of all, Section 2 summarizes the related work in the research area of NOMA and LA. In Section 3, we depict the configuration of the adopted system, and in Section 4, the optimization problem is formulated and analyzed. Section 5 details the proposed solution for the optimization problem. Numerical results are illustrated in Section 6, before we conclude the paper in Section 7.

2 The State of the Art

In this paper, we present a solution to user grouping and power allocation in NOMA systems through the use of LA, and specifically the OMA-based partitioning algorithms. Therefore, in this section, we present the state of the art of both NOMA and LA in relation to partitioning.

2.1 NOMA

Recently, NOMA technology has attracted a great attention and research effort [1,2,8]. Substantial research has been devoted to the field of NOMA through

[3] The notation for the paper is given below, so as to not distract from the content itself.

Table 1 Table of notations.

Notation	Description
h	Channel coefficient
$h_k,\ h_k(t),\ h_{n,k},\ h_{n,k}(t)$	h for U_k and $U_{n,k}$, and for U_k and $U_{n,k}$ at t
$\overline{h_k(t)},\ \overline{h_{n,k}(t)}$	The mean of the channel coefficient for U_k
K	Total number of users
N	Total number of groups
$\mathcal{K}$	Set of user indexes
$\mathcal{N}$	Set of group indexes
$\mathcal{G}$	Set of groups
g_n	Set of users inside the n-th group
$\lvert g_n \rvert$	Number of users inside g_n
$U_k,\ U_{n,k}$	User k and user k in group n
$g_n \backslash U_{n,k}$	The complementary set of users in set g_n
$\emptyset$	Empty set
$y_k,\ y_k(t)$	Signal from BS at U_k and U_k at time t
s_k	Transmitted signal intended for U_k
$P_n,\ P_n(t)$	Power budget for g_n, and P_n considering t
$\tau_{n,k}$	Indicator of whether U_k is in group n
Δ_t	Time period for considering average of h
S	Number of states per action
ϵ_k	Index of the current state of user k
$Q = (U_a, U_b)$	Input *query* of users to the EOMA
$f_c,\ f_d$	Carrier and Doppler frequency
W	Number of combinations
$p_{n,k},\ p_{n,k}(t)$	Allocated power for $U_{n,k}$ and considering t
$n_k,\ \sigma^2$	AWGN at U_k and Gaussian noise power
$\Gamma_k(t),\ \Gamma_{n,k},\ \Gamma_{n,k}(t)$	SINR of user U_k and $U_{n,k}$, and considering t
$I_{n,k}(t)$	Interference from other users to $U_{n,k}$ at t
b	Bandwidth of the channel
$R,\ R(t)$	Total data rate, and R considering t
$R_k, R_{n,k},\ R_{n,k}(t)$	Data rate of U_k and $U_{n,k}$, and considering t
R_{QoS}	Minimum required data rate for a user
B_K	The K-th Bell number
$\left\{ {K \atop \kappa} \right\}$	Stirling number of the second kind
L_c	Number of clusters
q_c	Set of users in cluster c
$\mathcal{C}$	Set of clusters
$\varphi_{c,k}$	Indicator of whether U_k is in cluster c
δ	Number of $\varphi_{c,k} = 1$ for a cluster
$U'_{c,g,k}$	User k in cluster c and group g in (13)
r_k	Rank
Υ_k	Ranking category of user k
E_c	Mean of channel fading in q_c
Θ_k	Cluster of U_k
γ	Precision in exhaustive search
v_U, v_L	Mobility factor of users and speed of light
α_c	Action in the LA for cluster c

the recent past, and user grouping and power allocation are among the many problems that have been researched.

A power allocation algorithm for NOMA networks was introduced in [9] so as to assure the fairness for users. Thereafter, in a single cell scenario, the

physical layer security was studied [10]. The sum-rate and outage probability for the downlink were analyzed in [11]. For the uplink, a power back-off method was investigated in [12]. The aforementioned research effort mainly focused on the intra-cell interference analysis. A dense multiple cell network for NOMA techniques considering inter-cell interference, where both uplink and downlink transmissions were evaluated, was studied in [13]. The study of the mmWave networks with NOMA was carried out in [14] and [15] with a focus on random beamforming without considering the locations of the users. Thereafter, the "the beamforming" strategy and power allocation coefficients were jointly optimized for maximizing the throughput [16]. In addition, stochastic geometry, which is able to characterize the communication distances between transceivers by providing a spatial framework, has also been utilized in NOMA [13,17] to model the locations of primary and secondary NOMA receivers.

In terms of user grouping and the corresponding ML techniques applied in NOMA, the study is still in its infancy. In [18], a dynamic method for classifying users for power allocation was investigated. The channel coefficients were sorted from high to low, and were assigned to channels, such that the difference between the users in each group increased. In [3], the authors utilized a K-means scheme for user grouping based on geolocation, where they considered the grouping of NOMA systems in, for example, school halls. Maximum weight matching was adopted in [2] to build non-disjoint groups per RB and for subsequently allocating power, given the obtained groups. In [19], proportional fairness (PF) was applied through an exhaustive search to allocate groups to the RBs. Groups of size two were investigated in [8] by invoking the Hungarian algorithm, and in [20] through PF for power allocation.

The above studies did not explore the stochastic nature of the wireless communication environment, and the channel coefficients for the different users were assumed to be known. Therefore, the optimization results based on the known channel coefficients were valid only when the coefficients were not far from reality. In practice, channel coefficients may vary along time rapidly, and channel sounding may not be frequent enough to capture the instantaneous change of mobile radio channels in the stochastic environment. In such situations, appropriate channel coefficients are to be used as the basis for user grouping and power allocation.

2.2 Learning Automata and Object Migration Automata

LA can be applied to solve many different problems, including OPPs in a random environment. The OPP is, in general, NP-hard, and a special case of the OPP, in which the number of objects in each group is equal, is the EPP. The benchmark solutions for the EPP have involved the classic field of LA [30], namely, the OMA and its variations. A comprehensive review of the previously proposed solutions for the OPP/EPP can be found in [6].

As the wireless communication system operates in a stochastic environment, and the NOMA technology involves multiple user groupings for SIC, it

Fig. 1 Groups are assigned to orthogonal resources. The users within a group employ NOMA.

is natural to apply the most recent RL-based solution for such an application in order to improve the system's performance in the stochastic environment. In this study, we confirm the potential of applying the OMA in optimizing the system's performance in the stochastic environment for the NOMA, which, to the best of our knowledge, has not been previously addressed in the literature.

3 System description

Consider a simplified single-carrier downlink cellular system that consists of one base station (BS) and K users that are to be divided into N groups for NOMA operation. User k is denoted by U_k where $k \in \mathcal{K} = \{1, 2, \ldots, K\}$. Similarly, the set of groups are denoted by $\mathcal{G} = \{g_n\}$, $n \in \mathcal{N} = \{1, 2, \ldots, N\}$, where g_n is the set of users inside the n-th group. Each user can only be in one of the groups implying that $g_n \cap g_o = \emptyset$ with $n \neq o$. When a user U_k belongs to group n, we use the notation $U_{n,k}$ to refer to this user and its corresponding group. The simplified notation U_k is adopted to refer to a user, when the group of the user is trivial or undetermined. For example, if we have 4 users in the system, we could have user 1 and user 3 belong to group 1, and user 2 and user 4 belong to group 2. In this case, when we want to refer to user 1 without its group, we use the notation U_1. Likewise, when we want to mention user 4 belonging to group 2, we apply $U_{2,4}$.

In this paper, we will consider the case of a single BS and with K users connected to that BS. NOMA is applied to each group, but different groups are assigned to orthogonal resources. One realization of such a communication scenario is shown in Fig. 1. In this scenario, the BS has assigned a frequency band to the K users. The users are to be grouped in N groups, each of which occupies a time slot. Here the orthogonal resource is the set of time slots, but it could just as well be other orthogonal resources, such as frequency bands. For mobility, the users are expected to move within a defined area. The users' behavior in a university or an office building are examples of where their behavior coincides with the mobility model utilized in this paper.

3.1 Channel Model

The channel model coefficient for U_k is denoted by $h_k(t)$ and refers to the channel fading between the BS and U_k along time. The channel coefficient is generated based on the well-recognized mobile channel model, which statistically follows a Rayleigh distribution [31]. The parameters of the channel

configuration will be detailed in the sections listing the numerical results. Note that our proposed LA solution can handle non-stationary stochastic processes, and so, it is distribution-independent. Therefore, the current Rayleigh distribution can be replaced by any other channel model, based on the application scenario and the environment.

3.2 Signal Model

In NOMA, each user in a group may suffer from both intra- and inter-group interference [3]. Intra-group interference refers to interference from other users within the same group, and inter-group interference refers to interference from users in other groups that employ the same band at the same time. In this study, we consider the case that different groups adopt orthogonal resources such that the inter-group interferences are non-existent.

Based on the NOMA protocol, the BS sends different messages to the users of a group in a single time slot via the same frequency band. Consequently, the received signal y_k at time t for U_k can be expressed as

$$y_k(t) = \sqrt{p_{n,k}} h_k(t) s_k + \sum_{e=1}^{|g_n|-1} \sqrt{p_{n,e}} h_k(t) s_e + n_k, \tag{1}$$

where e is the index of the users in the set $g_n \backslash U_{n,k}$, which is the complementary set of $U_{n,k}$ in g_n. $|g_n|$ returns the number of users in g_n. The received signal $y_k(t)$ has three parts, including the signal intended to $U_{n,k}$, the signal from all users other than $U_{n,k}$ in the same group, and the additive white Gaussian noise (AWGN), where $n_k \sim \mathcal{CN}(0, \sigma_k^2)$ [32]. The transmitted signal intended for $U_{n,k}$ and $U_{n,e}$ are given by s_k and $s_e \sim \mathcal{CN}(0, 1)$ respectively. $p_{n,k}$ is the allocated power for $U_{n,k}$. Further, the total power budget for group g_n is given by P_n.

The BS' signals are decoded at the users through the SIC by using the channel coefficients in an ascending order [2]. As a result, through SIC, a user with a good channel quality can remove the interference from the users possessing poor channel qualities, while users with poor channel qualities decode their signals without applying SIC[4]. Hence, for user $U_{n,k}$, successful SIC is applied when the following requirement fulfills,

$$|h_{n,w}(t)|^2 \leq |h_{n,k}(t)|^2, \tag{2}$$

where w is the index of the users that have lower channel coefficients than user k in the user group g_n. Note that the channel coefficient for a certain user may be quite different in distinct frequency levels. In this study, similar to assumptions in [33] and [34], we assume that the ranking of channel coefficients along time, on average, keep the same for all the users. Indeed, the differences

[4] We assume that differences between the time average of channel coefficients are due to the distinct geolocations of the users. Although the channel coefficient for a user may vary in different frequency bands, it is assumed that the ranking of the time averages of the coefficients among the various users maintains the same order in the different bands due to their distinct geolocations. With this assumption, the heterogeneity in different frequency bands will not influence the results of the user grouping.

of channel coefficients are due to the various distances among the users to the BS, which makes the assumption in this paper valid. Eq. (2) implies that $U_{n,k}$ is able to remove the messages to the users with lower channel coefficients in its group via SIC and then retrieve its own message. Hence, $U_{n,k}$ considers users with higher channel coefficients in its group as interference in the decoding process.

4 Problem Formulation

In this section, we formulate the problem to be solved. The first problem, which we refer to as the main problem, is formulated when instantaneous channel coefficients, say, at time t_0, are considered for grouping and power allocation. Here time t_0 is the time instant when the channel sounding is employed for the current round of user grouping and power allocation. Ideally, if the optimization of user grouping and power allocation is quick enough, and if the packet is short enough, the channel coefficients can be considered as a constant. In addition, if it is possible to re-group users and allocate power to them more often than the changes of channel coefficients, the system becomes adaptive and can thus be operated always in an optimized manner.

In reality, though, by studying the complexity of the main problem, we show that it is computationally very costly to solve. Therefore, it is not practical for the BS to follow the instantaneous channel condition as the basis for user grouping. For this reason, the main problem is divided into two sub-problems, where in the first sub-problem, the user channels are clustered based on the time averages of the coefficients, and subsequently, in the second sub-problem, the power allocation is considered.

4.1 Problem Formulation of Overall System

The objective of the main problem is to maximize the overall data rate given channel coefficients $h_k(t_0)$ and power budget. To take advantage of NOMA, the K users need to be divided into N groups, and for each group, the users share the same resource block (in time and frequency). Additionally, power allocation is to be optimized according to the channel conditions for each user in that group. To ease in the formulation, without loss of generality, we assume that the channel coefficients are sorted in ascending order, i.e., $h_1(t_0) \leq h_2(t_0) \leq \ldots \leq h_K(t_0)$[5].

Following the description given in [3], for the n-th group, after deployment of SIC, the SINR (signal-to-interference-plus-noise ratio) of user k in g_n is given by

$$\Gamma_{n,k}(t_0) = \frac{p_{n,k}|h_{n,k}(t_0)|^2}{I_{n,k}(t_0) + \sigma^2}, \tag{3}$$

[5] This assumption applies only in the problem formulation with instantaneous channel coefficients at t_0. Understandably, the channel coefficients will change along time due to the stochastic behavior, and the ranking of instantaneous channel coefficients belonging to different users may change from time to time due to channel fading.

where $p_{n,k}$ is the power allocated to $U_{n,k}$. Clearly, it is true that for any group n, $\sum_{j,\forall U_j \in g_n} p_{n,j} \leq P_n$ holds, where P_n denotes the power budget of g_n, with σ^2 denoting the Gaussian noise power. Parameter $I_{n,k}(t)$ represents the intra-group interference from other users to $U_{n,k}$, as

$$I_{n,k}(t_0) = \sum_{\substack{j, \\ \forall j>k,\ \{U_j,\ U_k\} \in g_n}} |h_k(t_0)|^2 p_{n,j}. \tag{4}$$

Given that the SIC requirement is fulfilled, the achievable data rate of user k is

$$R_{n,k}(t_0) = b \log_2 \left(1 + \Gamma_{n,k}(t_0)\right), \tag{5}$$

where b is the bandwidth of the channel. Therefore, the total achievable data rate for the system is expressed as

$$R(t_0) = \sum_{k=1}^{K} b \log_2 \left(1 + \Gamma_{n,k}(t_0)\right). \tag{6}$$

Based on the above analysis, we can formulate the optimization problem as follows.

$$\max_{\{g_n\},\{p_{n,k}\}} \quad R(t_0) \tag{7a}$$

$$\text{s.t.} \quad g_n \cap g_o = \emptyset, \quad n \neq o, \quad n, o \in \mathcal{N} \tag{7b}$$

$$\sum_{j,\forall U_j \in g_n} p_{n,j} \leq P_n, \quad \forall n \in \mathcal{N}, \tag{7c}$$

$$R_{n,k}(t_0) \geq R_{QoS}, \quad k \in \mathcal{K}, \tag{7d}$$

$$h_i(t_0) > h_j(t_0), \quad \forall i > j, \quad i, j \in \mathcal{K}. \tag{7e}$$

We now explain the significance of each of the above equations. The constraints in (7b) say that each user can only be part of one group. In (7c), we state that the sum of powers for a certain group needs to be less than or equal to P_n, which guarantees that the total power of the group is within the power constraint. The constraints in (7d) concern the demands on the Quality of Service (QoS) for each user. Hence, the data rate of a user needs to be above a specified required value, R_{QoS}, ensuring the QoS to all users, and addressing the fairness issue [32]. Finally, Eq. (7e) addresses the SIC requirement.

4.2 Complexity Considerations

We shall now consider the complexity of the associated problem. Considering the users and their placement into different groups, the minimum number of combinations possible is a Bell number, without even reckoning with the task of power allocation, where the Bell number gives the number of possible partitions of a set. In our case, we have B_K options where B_K is the K-th Bell number. Our task is to partition K users into κ non-empty sets. Considering that κ, without any additional constraints, can range from 1 to K, the consecutive Bell number for K is given by

$$B_K = \sum_{\kappa=1}^{K} \left\{ {K \atop \kappa} \right\}, \tag{8}$$

where $\left\{ {K \atop \kappa} \right\}$ is the Stirling numbers of the second kind [35]. Consequently, it follows that

$$\left(\frac{K}{e \ln K}\right)^K < B_K < \left(\frac{K}{e^{1-\lambda} \ln K}\right)^K, \tag{9}$$

which is exponential with regard to K [35], and $\lambda > 0$. Because the number of possible combinations to solve such a maximization problem (as stated in (7)) increases drastically with the number of users, and since the power allocation problem is non-convex [3], finding an optimal solution to the problem, based on instantaneous $h_k(t_0)$, is computationally hard. Further, when the system is to be adaptive to the changes in the environment, computations are to be carried out very frequently. Therefore, it is more practical to aim for a compromised solution, where the problem is divided into two sub-problems.

Specifically, in the first problem, we cluster the users into categories based on the time averages of the channel coefficients, and in the second step, we group the users based on the obtained categories, and proceed to solve the power allocation phase. The rationale behind such a computational paradigm is that we capture the long time average of the channel coefficients for the purpose of *grouping*, and thereafter, the power allocation is based on the available instantaneous values of h, or a time average computed over a certain number of channel sounding samples of h. Thus, as the grouping is considered more costly than the power allocation, by doing the grouping based on the mean, and being able to do power allocation more often at a minimal cost once the groups have been established, the computational effort is kept low.

Since there is no known solution for the general OPP, we further simplify the model for the solution, to consider the equi-partitioning of the users, namely the EPP. When all the groups are of equal sizes, we have the combination number W as:

$$W = \frac{K!}{\left(\frac{K}{N}!\right)^N N!}, \tag{10}$$

where $\frac{K}{N}$ is an integer. As a result of the above, we observe that the partitioning problems are still characterized by a combinatorial issue, but the number is significantly smaller than the Bell numbers. Note that the problem is still more complicated than just finding an optimal partitioning once and for all, because the environment changes stochastically.

4.3 Problem Formulation of Clustering

For NOMA, the channel coefficients of the users in a group need to be as different as possible to attain to a successful NOMA operation. To group the users with different coefficients, we first want to cluster the users with similar channel coefficients, and then select a single user from each cluster in order to formulate the groups. In other words, we want to cluster similar users together first, and then group them by selecting one user from each cluster. The first problem is the clustering problem, which is formulated in this subsection. The problem for user grouping, together with power allocation, is formulated in the next subsection.

The criterion for user clustering involves the *time averages* of the channel coefficients, denoted by $\overline{h_k(t)}$. In other words, the users that have similar average channel coefficients will be clustered together. The reason behind this is that the task of grouping the users is relatively costly in terms of computation, and it is, further, not cost-efficient to apply it based on the channel's instantaneous values. If we cluster the users according to the time averages, we can efficiently reduce the computational cost, and at the same time, capture the advantages of statistically employing NOMA.

In this study, as mentioned, we consider clustering users to groups of the same size. Therefore the number of the clusters is $L_c = \frac{K}{N}$, where L_c and N are integers[6]. Let q_c be the set of users in cluster c, where $c \in \mathcal{C} = \{1, \ldots, L_c\}$ is the index of the set of clusters. Clearly, for the clustering problem, the differences between the coefficients in each cluster needs to be minimized, and the problem can be formulated as

$$\min_{\{\varphi_{c,k}\}} \quad \sum_{c=1}^{L_c} \sum_{k=1}^{K} \varphi_{c,k} |\overline{h_k(t)} - E_c|, \tag{11a}$$

$$\text{s.t.} \quad \sum_{c=1}^{L_c} \sum_{k=1}^{K} \varphi_{c,k} = K, \quad c \in \mathcal{C}, k \in K, \tag{11b}$$

$$\sum_{k=1}^{K} \varphi_{c,k} = N, \quad \forall c, \tag{11c}$$

where $\varphi_{c,k}$ is an indicator showing the relationship between the users and the clusters, and is given by

$$\varphi_{c,k} = \begin{cases} 1, \text{when } U_k \text{ belongs to cluster } c. \\ 0, \text{otherwise.} \end{cases} \tag{12}$$

Additionally, the mean value of the channel fading in each cluster is denoted by the parameters E_c and δ, which are given by $E_c = \frac{1}{\delta} \sum_{k=1}^{K} \overline{h_k(t)} \varphi_{c,k}$, and $\delta = \sum_{k=1}^{K} \varphi_{c,k}$ respectively, where the objective function is stated in Eq. (11a). Specifically, for all the clusters, we want to minimize the difference of the channel coefficients between the users inside each of them. Eqs. (11b) and (11c) give a description of the variable $\varphi_{c,k}$ for user k in c. Hence, the sum of the variable $\varphi_{c,k}$ needs to be equal to the number of users, implying that all the users need to be a part of one cluster, and in each cluster, there needs to be an equal number of users.

[6] In reality, if L_c is not an integer, we can add dummy users to the system to satisfy the requirement. *Dummy users* are virtual users that are not part of the real network scenario, but are needed for constituting an equal size for all the clusters. Thus, the dummy users are used for the clustering, but these users are not real, and no resources are given to them in the power allocation process (they should be excluded in the power allocation process).

The result of the clustering problem, i.e., the $\{\varphi_{c,k}\}$ that minimizes the objective function, can be re-arranged in an $L_c \times N$ matrix, as

$$\begin{array}{c} \\ 1 \\ 2 \\ \dots \\ L_c \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & N \end{array} \\ \begin{bmatrix} U'_{1,1,k} & U'_{1,2,k} & \dots & U'_{1,N,k} \\ U'_{2,1,k} & U'_{2,2,k} & \dots & U'_{2,N,k} \\ \dots & \dots & \dots & \dots \\ U'_{L_c,1,k} & U'_{L_c,2,k} & \dots & U'_{L_c,N,k} \end{bmatrix} \end{array}, \tag{13}$$

indicating L_c clusters with N users in each cluster. The user in the matrix is indexed by $U'_{c,g,k}$, where c is the index of the cluster, and g is the index of a user in a certain group, while k indicates the user (the position of a user k, in the matrix, is an independent term). Note that $U_{n,k}$ is different from $U'_{c,g,k}$ as the indexes are different.

For the next step, the problem is then to group users together by selecting one user from each cluster, and to then allocate power to them in order to achieve the maximized sum of data rate.

4.4 Problem Formulation of Power Allocation

As the grouping and power allocation tasks are inter-twined, we must consider these two aspects jointly when we study the maximization of the data rate. Although from the first step, we have obtained information on which users have similar channel coefficients, we observe that the power allocation problem remains unresolved. Therefore, we need to consider the power allocation of the users inside each cluster so as to be able to solve both the grouping and power allocation in NOMA.

Clearly, from the output of the clustering, we know which users are similar, and when we take one user from each cluster and construct N groups, the size of each group will be L_c. Without loss of generality, we can assume that the average channel coefficients are sorted in ascending order, i.e., $\overline{h_1(t)} \leq \overline{h_2(t)} \leq \dots \leq \overline{h_K(t)}$ (similar to [33] and [34]). If we now consider user grouping and power allocation based on the average channel coefficients, the problem can be formulated as

$$\max_{\{g_n\},\{P_n\}} \quad R \tag{14a}$$

$$\text{s.t.} \quad g_n \cap g_o = \emptyset, \quad n \neq o, \quad n, o \in \mathcal{N}, \tag{14b}$$

$$\sum_{j,\forall U_j \in g_n} p_{n,j} \leq P_n, \quad n \in \mathcal{N}, \tag{14c}$$

$$R_{n,j}(t) \geq R_{QoS}, \quad j \in \mathcal{K}, n \in \mathcal{N}, \tag{14d}$$

$$\overline{h_i(t)} > \overline{h_j(t)}, \quad \forall i > j, \quad i, j \in \mathcal{K}, \tag{14e}$$

$$|g_n \cap q_c| = 1, \quad \forall c, \ \forall n, \tag{14f}$$

$$\sum_{j,\forall U_j \in g_n} \tau_{n,j} = L_c, \quad \forall n, \tag{14g}$$

$$\sum_{n,\forall n \in \mathcal{N}} \sum_{j,\forall U_j \in g_n} \tau_{n,j} = NL_c. \tag{14h}$$

In Eq. (14a), the parameter R is calculated based on Eq. (6) when $h_k(t)$ is replaced by $\overline{h_k(t)}$, indicating that this rate is based on the average channel coefficients. Further, in (14b), we state that the groups need to be disjoint. Hence, any user can only be in a single group. In Eq. (14c), we address the constraint for the power budget. The QoS constraint is given in (14d), which ensures the fairness among the users. The SIC constraint is given in Eq. (14e). The constraint in Eq. (14f) specifies that only a single user is selected to formulate a group from each cluster. Finally, in Eq. (14g), we introduce an indicator $\tau_{n,k}$, stating whether U_k is in group n, as

$$\tau_{n,k} = \begin{cases} 1, \text{when } U_k \text{ belongs to group } n. \\ 0, \text{otherwise.} \end{cases} \tag{15}$$

This constraint states that each group has L_c users. Furthermore, all users should belong to a certain group, which is given in Eq. (14h).

The solution to this problem will provide the grouping of users along with their corresponding power allocations. Note that the power allocation is calculated based on the average of the channel coefficients. When communication happens, these coefficients may be different. Thus, the power allocation can be done for time averages of the channel sounding, or for instantaneous values.

Comparing the problem in Eq. (7) with the sub-problems in Eqs. (11) and (14), we record the following differences. (a) In Eq. (7), the instantaneous channel coefficients are followed for grouping and optimization. However, in Eqs. (11) and (14), the clustering and grouping of users are based on the respective time-averaged values. (b) In Eq. (7), the groups of users may have different sizes, while in Eqs. (11) and (14), the sizes of all the groups are equal. Indeed, since following the instantaneous channel coefficients is both costly and impractical, the task of following the average values becomes a reasonable and feasible foundation for the grouping. Considering that the equi-partitioning of the users reduces the complexity of the original problem, practical and adaptive solutions based on RL algorithms can be proposed. In the next section, we will propose a two-step solution corresponding to the sub-problems, based on the adaptive Tabula rasa RL paradigm.

5 Solution to User Grouping and Power Allocation

The problem of grouping and power allocation in NOMA systems is two-pronged. Therefore, in Section 5.1, we only consider the first issue of the two, namely the clustering and the grouping of the users. We will show that our solution can handle the stochastic nature of the channel coefficients of the users, while it is also able to follow changes in their channel behaviors over time, ensuring that the system can prudently follow the nature of the channels that are similar to what we will expect in a real system. More specifically, we will categorize users into clusters based on similar channel coefficients for long-term fading. Because the values of h for different users are stochastic, we need a method for capturing the long-time average of the channels. Therefore, we

exploit a ML algorithm from within the OMA family for clustering them. Specifically, we utilize the Enhanced Object Migration Automata (EOMA) to capture the users' similarities, for the purposes of categorization. Thereafter, as mentioned above, the users are grouped by taking a single user from each cluster so that users within any one group have distinct channel coefficients. Once the groups have been established in Section 5.1, we can utilize these groups to allocate power among them either instantaneously or for a time interval using heuristic-based solutions.

5.1 Clustering Through EOMA

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 while the computational complexity remains low [36]. OMA algorithms are part of the LA paradigm, and can solve partitioning problems by having LA instances cooperate to find the best partitioning [37]. With some insight, we see that the family of OMA algorithms are based on Tabula rasa RL. Without any prior knowledge of the system parameters, channels, or clusters in our case, the OMA self-learns by observing the environment that it interacts with, over time. For our problem, the communication system constitutes the environment, which can be observed by the OMA through, e.g., channel sounding. By gaining knowledge from the system behavior and improving incrementally through each interaction with the environment, the OMA algorithms prove themselves to be compelling mechanisms for solving complex and stochastic problems. The OMA algorithms attempt to learn the "true partitioning" in different grouping scenarios, based on the elements from the respective groups that are accessed together. The "true partitioning" that occurs in nature is always unknown, but assuming that the true partitioning consists of equi-sized clusters, the OMA algorithms can find these with a high accuracy [6], [7], [38].

In the OMA, the users of our system need to be represented as abstract objects. Therefore, in OMA terms, the users are called objects. The OMA algorithms require a number of states per action, indicated by S. An action in LA is a solution that the algorithm can converge to. In our system, the actions are the different clusters that objects can be assigned to. Hence, based on the current state of an object, we know that object's action, which is precisely its current cluster in the system. Therefore, each object, or user in our case, has a given state indicated by $\epsilon_k = \{1, 2, ..., SL_c\}$, where ϵ_k denotes the current state of U_k, S is the number of states per action, and L_c is the number of clusters. Clearly, because we have Lc clusters, the total number of possible states is SL_c. To indicate the set of users inside cluster c, where $c \in [1, 2, \ldots, L_c]$, we have q_c. The cluster for a given user, k, is represented by Θ_k, where the set of clusters is denoted by $\mathcal{C}$ and $\Theta_k \in \mathcal{C} = \{q_1, q_2, \ldots, q_{L_c}\}$.

The states are the foundational memory components of the OMA algorithms, and the objects are moved in and out of states as they are penalized or rewarded in the RL process. In this paper, we utilize the EOMA variant

of OMA, where each object has an equal number of possible states. We say that the algorithm has converged when all the objects have reached the two innermost states of an action. When convergence has been attained, we reckon that the solution that the EOMA algorithm has found, is sufficiently accurate.

The requirement for convergence can be tuned through the number of states introduced to the system. Consequently, introducing a deeper state space increases the solution's accuracy of finding the "true partitioning". However, the time before convergence is achieved, also increases with the number of states. Therefore, the state depth, given by S, is a trade-off between time and accuracy. In Fig. 2, we describe the state numbering of any two actions. By way of example, consider a scenario with three possible clusters: the first cluster of the EOMA will have the state numbering from 1 to S, where the innermost state is 1, and the second innermost state is 2, while the boundary state is S. For the second cluster, it will have the innermost state $S+1$ and the second innermost state $S+2$, while the boundary state is $2S$. Likewise, for the third cluster, the numbering will be $2S+1$ for the innermost and $2S+2$ for the second innermost state whilst $3S$ for the boundary state.

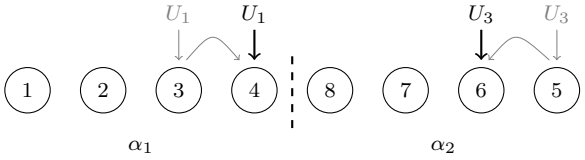


Fig. 2 An example of the states of the EOMA, for the updates for the penalty for U_1 and U_3 when they have the same ranking category.

Algorithm 1 Clustering of Users

Require: $\overline{h_k(t)}$ for all users K over Δ_t
  while not converged **do** {Convergence is reached when all users are in the two most internal states of any action}
    for all K **do**
      Rank the users from 1 to K {Index 1 is given to the user with lowest $\overline{h_k(t)}$ (K to the highest)}
    end for
    for $\frac{K}{N}$ pairs (U_a, U_b) of K **do** {The pairs are chosen uniformly from all possible pairs}
      if $\Upsilon_a = \Upsilon_b$ **then** {If U_a and U_b have the same ranking category}
        if $\Theta_a = \Theta_b$ **then** {If U_a and U_b are clustered together in the EOMA}
          Process Reward
        else {If U_a and U_b are not clustered together in the EOMA}
          Process Penalty
        end if
      end if
    end for
  end while{Convergence has been reached}

Algorithm 1 gives the overall operation for the clustering of the users. The functionalities on receiving a reward or a penalty, as the EOMA is interacting with the NOMA system, are given in Algorithms 2 and 3. In the algorithms, we consider the operation in relation to a pair of two users U_a and U_b ($Q = (U_a, U_b)$). The EOMA considers users in pairs (called *queries*, denoted by Q). Through information about their pairwise rankings, we work towards a clustering of the users into the different channel categories. For each time instant Δ_t, the BS obtains values of $h_k(t)$ through channel sounding, and we use the average of the samples of Δ_t as input to the EOMA ($\overline{h_k(t)}$). Note that when $\Delta_t = 1$, the instantaneous values are utilized as the input.

The BS proceeds to rank the users, indicated by $r_k = \{1, 2, ..., K\}$, where each U_k is given a single value of r_k for each Δt. For the ranks, $r_k = 1$ is given to the user that has the lowest channel coefficient compared to the total number of users; $r_k = K$ is given to the user with the highest channel coefficient of the users, and the others are filled in between them with ranks from worst to best. Furthermore, the values of these ranks corresponds to ranking categories, denoted by Υ_k for U_k, where $\Upsilon_k = \{r \in [1, N] = 1, r \in [1 + N, 2N] = 2, r \in [1 + 2N, 3N] = 3, \ldots, r \in [K - N + 1, K] = L_c\}$. In this way, even if the users have similar channel conditions, they will be compared, and the solution can work on determining the current best categorization of the K users for the given communication scenario. As depicted in Algorithm 1, we check the users' ranking categories in a pairwise manner. If the users in a pair (query) are in the same ranking category, they will be sent as a query to the EOMA algorithm. The EOMA algorithm will then work on putting the users that are queried together in the same cluster, which, in the end, will yield clusters of users with similar channel coefficients. More precisely, if two users have the same ranking categories, they are sent as a query to the EOMA and the LA is rewarded if these two users are clustered together, and penalized if they are not.

Algorithm 2 Process Reward

Require: $Q = (U_a, U_b)$ {A query (Q), consisting of U_a and U_b}
Require: The state of U_a (ϵ_a) and U_b (ϵ_b)
  if $\epsilon_a \mod S \neq 1$ **then** {U_a not in innermost state}
    $\epsilon_a = \epsilon_a - 1$ {Move U_a towards innermost state}
  end if
  if $\epsilon_b \mod S \neq 1$ **then** {U_b not in innermost state}
    $\epsilon_b = \epsilon_b - 1$ {Move U_b towards innermost state}
  end if
  return The next states of U_a and U_b

As an example, let us consider four users in a NOMA communications scenario for $\Delta_t = 5$. The users should be grouped into two groups. Therefore, we need to categorize them into two clusters based on their channel conditions: one with weak channel conditions (ground truth in this example: $\Upsilon_1 = 1$ and $\Upsilon_2 = 1$) and the other with strong channel conditions (ground truth in this

Algorithm 3 Process Penalty

Require: $Q = (U_a, U_b)$ {A query (Q), consisting of U_a and U_b}
Require: The state of U_a (ϵ_a) and U_b (ϵ_b)
  if $\epsilon_a \mod S \neq 0$ and $\epsilon_b \mod S \neq 0$ **then** {Neither of the users are in boundary states}
    $\epsilon_a = \epsilon_a + 1$ {Move U_a towards boundary state}
    $\epsilon_b = \epsilon_b + 1$ {Move U_b towards boundary state}
  else if $\epsilon_a \mod S \neq 0$ and $\epsilon_b \mod S = 0$ **then** {U_b in boundary state and U_a not in boundary state}
    $\epsilon_a = \epsilon_a + 1$
    $temp = \epsilon_b$
    x = unaccessed user in cluster of U_a closest to boundary state
    $\epsilon_x = temp$
    $\epsilon_b = \epsilon_a$
  else if $\epsilon_b \mod S \neq 0$ and $\epsilon_a \mod S = 0$ **then** {U_a in boundary state and U_b not in boundary state}
    $\epsilon_b = \epsilon_b + 1$
    $temp = \epsilon_a$
    x = unaccessed user in cluster of U_b closest to boundary state
    $\epsilon_x = temp$
    $\epsilon_a = \epsilon_b$
  else {Both users are in boundary states}
    $\epsilon_y = \epsilon_{\{a \ or \ b\}}$ {y equals a or b with equal probability}
    $temp = \epsilon_y$
    x = unaccessed user in cluster of $U_{\not y}$ closest to boundary state
    $\epsilon_x = temp$
    $\epsilon_y = \epsilon_{\not y}$ {Move chosen user (y) to cluster of $\not y$}
  end if
  return The next states of U_a and U_b

example: $\Upsilon_3 = 2$ and $\Upsilon_4 = 2$). First, when $h_k(5)$ for the different users are obtained, we rank them according to $\overline{h_k(5)}$. Then, we consider the arbitrary pair $Q = (U_1, U_3)$, ranked $r_1 = 2$ giving $\Upsilon_1 = 1$ and $r_3 = 2$ giving $\Upsilon_3 = 1$ ($r = \{3, 4\}$ resulting in $\Upsilon = 2$). Additionally, their current states are $\epsilon_1 = 3$ and $\epsilon_3 = 5$. For this scenario the state depth for each action is four, meaning that we have 8 states in total ($SL_c = 8$). Following the descriptions given in [6], [7], [39], or the same concept that we observe in Algorithms 2 and 3, we understand that the objects are currently not clustered together. Therefore, we will penalize them according to Algorithm 3. A visualization of the example is depicted in Fig. 2.

When the algorithm has converged, the users in distinct actions constitute different clusters. We will then invoke Algorithm 4, to obtain the groups that are needed for the power allocation, where the users are selected into groups based on their ranking of $\overline{h_k(t)}$ within each cluster. Again we use the ranking information of the users, where users with the same rank in each cluster are put together. Thus, all the users with the same rank in each of their respective clusters, will form a group.

Algorithm 4 Get Groups

Require: The users and information about their cluster obtained by the EOMA
  for all clusters in $\mathcal{C}$ **do**
    Rank the users from 1 to Lc based on $\overline{h(t)}$ {$r = 1$ to the user with lowest value ($r = Lc$ to the highest)}
  end for
  for Number of groups (N) **do**
    for all r **do**
      for all clusters in $\mathcal{C}$ **do** {Each group will consist of one user from each cluster with the same rank}
        g_n = One user from each cluster with rank r
      end for
    end for
  end for
  return The users' groups

5.2 Power Allocation via Heuristic-based Solutions

Once the grouping of the users has been established, we can allocate power to the different users in such a way that the joint data rate (R) is maximized. For a group with $\frac{K}{N} = L_c$ users and power budget P_n, the problem can be expressed by:

$$\max \sum_{k=1}^{K} b \log_2 \left(1 + \Gamma_{n,k}\right), \tag{16a}$$

$$\text{s.t.} \sum_{k=1}^{K} p_{n,k} \leq P_n, \quad \forall n, n \in \mathcal{N}, \tag{16b}$$

$$0 \leq p_{n,k}, \quad \forall n, n \in \mathcal{N}, \forall k, k \in \mathcal{K}, \tag{16c}$$

$$R_{Qos} \leq R_k, \quad \forall k, k \in \mathcal{K}, \tag{16d}$$

where $\Gamma_{n,k} = \frac{p_{n,k}|h_{n,k}|^2}{|h_{n,k}|^2(P_n - \sum_{\forall i \leq k} p_{n,i}) + \sigma^2}$. Our goal is to determine the power to the different users within each group so as to maximize the total data rate of all the groups.

Note the the objective function for the optimization may be changed to also work with other functions, such as that of maximizing the minimum data rate within a group. There are also numerous ways of power allocation in various communication scenarios [40],[32]. The power allocation schemes can be replaced by any other algorithm and will not change the nature of the RL procedure. However, in this paper, we will consider two heuristic-based algorithms, namely, the greedy algorithm and the channel coefficient based algorithm for maximizing the sum rate.

For the greedy solution, we allocate to the users with limited power to just fulfill the minimum required data rate, and give the remaining power to the user with the best channel coefficient, as presented in Algorithm 5. Given Eqs. (5) and (6), allocating the majority of power to the users with higher values of h will result in a higher sum rate for the system. Consequently, the stronger users are benefited more from the greedy solution than those with

weaker channel coefficients. Nevertheless, the weak users' required data rate is ensured, and can be adjusted to the given scenario. The greedy solution can also be used for checking the feasibility. When all users are given power values that are just sufficient to fulfill the QoS requirement, and if the total power is sufficient, we deem the solution obtained as being "feasible".

The main drawback of the greedy solution is that the data rates among the users may be highly unbalanced. To mitigate this drawback, we propose another solution based on a relation between the values of $|h|^2$ of the different users, when they are a part of an established group. This solution is depicted in Algorithm 6. As observed in the algorithm, we base a linear algebraic system on the SINR (Eq. (3)), intra-group interference (Eq. (4)) and the data rate (Eq. (5)) for optimizing the sum rate of the system given by Eq. (6). Firstly, the data rate of the user with the weakest h is ensured by setting its data rate equal to R_{QoS}. Once we know the data rate of the weakest user, we can calculate the power that needs to be given to that particular user. Consequently, we then find the power for the users between the weakest and strongest users for the given group (based on $h_{n,i}$), where we use the SINR of the previous user times the relation between h-values for the previous user and the user for whom we are finding the SINR. Once we have the SINR for the user in-between the weakest and strongest, we can calculate its needed power based on Eq. (3). The strongest user is then given the remaining power by subtracting the power allocated for the other users from the total power budget of its group. The process is repeated for all the groups.

6 Numerical Results

For the experiments reported here, we used MatLab for simulating the values of h. Additionally, a Python script was utilized for simulating the LA solution to the user grouping and the greedy solution to power allocation.

Algorithm 5 Greedy solution for the power allocation

Require: $h_{n,k}$ for all users K {Requires the value of $h_{n,k}$ for t $\left(h_{n,k}(t)\right)$ or Δ_t $\left(\overline{h_{n,k}(\Delta_t)}\right)$}
Require: R_{QoS} {The minimum required data rate}
Require: $\mathcal{G}$ {The groups established in Algorithm 4}
  for all g_n, in $\mathcal{G}$ **do**
    for all users, i from 1 to the size of g_n **do** {Ordered, where user 1 has the lowest h (L_c has the highest h)}
      Solve for $p_{n,i}$ using $R_{QoS} = B\log_2\left(1+\frac{p_{n,i}|h_{n,i}|^2}{|h_{n,i}|^2(P_n-\sum_{\forall k\leq i}p_{n,k})+\sigma^2}\right)$ {The feasibility check}
    end for
    if $P_n - \sum_i p_{n,i} \geq 0$ **then**
      $p_{n,Lc} = P_n - \sum_{j,\forall p_{n,j}\in g_n \setminus p_{n,L_c}} p_{n,j}$ {The problem is feasible, and we give the remaining power to the strongest user}
    end if
  end for

Algorithm 6 Channel-coefficient based solution for power allocation

Require: $h_{n,k}$ for all users K {Requires the value of $h_{n,k}$ for t $\left(h_{n,k}(t)\right)$ or Δ_t $\left(\overline{h_{n,k}(\Delta_t)}\right)$}
Require: R_{QoS} {The minimum required data rate}
Require: $\mathcal{G}$ {The groups established in Algorithm 4}
  for all g_n, in $\mathcal{G}$ **do**
    for all users, i, in g_n **do** {Ordered, where user 1 has the lowest h (L_c has the highest h)}
      if i is 1 **then** {For the weakest user}
        $R_{n,1} = R_{QoS}$ {Data rate of weakest user}
        Solve for $p_{n,1}$ using $R_{n,1} = B\log_2\left(1 + \frac{p_{n,1}|h_{n,1}|^2}{|h_{n,1}|^2(P_n - p_{n,1}) + \sigma^2}\right)$
      else if i is L_c **then** {For the strongest user}
        $p_{n,Lc} = P_n - \sum_{j,\forall p_{n,j} \in g_n \setminus p_{n,L_c}} p_{n,j}$ {We give the remaining power to the strongest user}
      else {For the user between the weakest and strongest user}
        $\Gamma_{n,i} = \frac{p_{n,i}|h_{n,i}|^2}{|h_{n,i}|^2(P_n - \sum_{\forall k \leq i} p_{n,k}) + \sigma^2}$
        Solve for $p_{n,i}$ using $\frac{\Gamma_{n,i}}{\Gamma_{n,i-1}} = \frac{|h_{n,i}|^2}{|h_{n,i-1}|^2}$ {SINR of user i is based on a relation between U_i and U_{i-1}}
      end if
    end for
  end for
  return The power for the different users in the different groups

The numerical results for the power allocation solution are based on the results obtained from the EOMA clustering and grouping. For the simulations, we used a carrier frequency of $5.7GHz$ and an underlying Rayleigh distribution for the corresponding values of $h(t)$. For mobility in our model, we utilized a moving pace corresponding to the movement inside an office building. Thus, we assumed a mobility factor of $2\,\frac{km}{h} = v_U$ for the users' receivers. We sampled the values of h according to $\frac{1}{2f_d}$, where f_d is the Doppler frequency, where the latter is expressed as $f_d = f_c(\frac{v_U}{v_L})$, and v_L is the speed of light.

In the figures given below, we use "Sample Number" (Δ_t) as the notation on the X-axis. The numerical results for the sub-problems will be presented separately in Sections 6.1 and 6.2.

6.1 Results for Grouping

For evaluating the simulation of clustering and grouping, we base our accuracy on whether or not the LA found the clusters that correspond to the minimized difference between the users in a cluster, based on the users' given mean values of h in the simulations. Remarkably, if it is provided with such pairwise inputs, the EOMA yielded a 100% accuracy in which the learned clustering was identical to the unknown underlying clustering in every single run for the example provided with $-30dB$ difference between values of h, and for groups of size $4, 6, 8, 10, 12, 14, 16, 18$ and 20, where the number of users in a group was equal to two. The difference between the users can be replaced by any

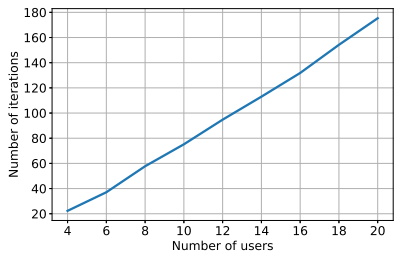


Fig. 3 Graph showing the average number of required iterations (Δ_t) before convergence is reached for different number of users, and groups of size two, based on the average of 100 independent experiments.

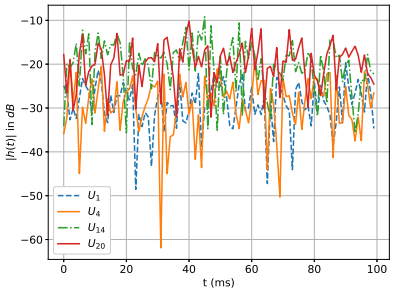


Fig. 4 Example of the simulated $|h(t)|$ for four different users in dB scale.

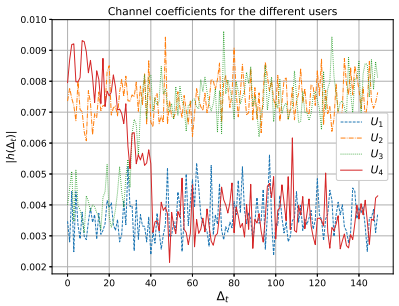


Fig. 5 Graph showing $|h(\Delta_t)|$ as a function of time Δ_t, where U_3 and U_4 changes distinctly around $40\Delta_t$.

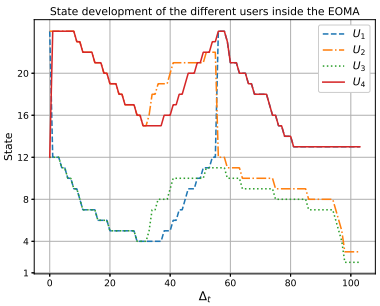


Fig. 6 The changes of states in the EOMA for different users over Δ_t, where the LA starts changing clusters around $40\Delta_t$.

other equivalent condition, and these values are only generated for testing the solution. The reader should observe that in a real-life scenario, the "true partitioning" is always unknown. The number of iterations that it took for the EOMA to achieve 100% accuracy for the different numbers of users is depicted in Fig. 3. These results were based on h values that are shown in Fig. 4. In the interest of simplicity of presentation, the plot shown in Fig. 4 is for 4 users. For the case of 20 users, the lines become hard to distinguish, even though the principles used in the simulation of h are the same. Notably, the EOMA retains its accuracy as the number of users increases, and yields 100% accuracy both for four users as well as 20 users[7].

When we formulated the problem in Section 3, for conciseness, we assumed that the ranking of the average values of the channel coefficients were kept the

[7] In these simulations, we used $S = 8$. The way that we obtained the solution's accuracy was in terms of whether or not the EOMA found the clusters that it should have found, based on the mean values of the different users in the NOMA system.

same. In fact, the proposed EOMA algorithm can follow the changes adaptively even if the mean values vary along time. Here in Figs. 5 and 6, we demonstrate that the EOMA is able to follow the changes adaptively when the users' channel coefficients vary along time. As depicted in Fig. 5, a change in channel coefficients happens at around $\Delta_t \approx 40$. From Fig. 6, we can observe that the EOMA quickly detects the change even as early as around $\Delta_t \approx 40$, and updates its clustering after approximately 20 samples. More specifically, we can observe from Fig. 5 that the most similar users are initially $U1$ and $U3$, and $U2$ and $U4$, which change to $U1$ and $U4$, and $U2$ and $U3$ after around 40 samples. To show the updates of the states in the EOMA, the states are depicted as a function of time in Fig. 6. Initially, the objects are randomly located in boundary states (State 24 and State 12 in this figure). We can observe that the users with similar coefficients move to the same cluster after a few interactions, and they eventually move to deeper states. When the environment changes at around time instant 40, they move shallower instead of deeper. Eventually, the user clusters are updated in accordance to the new environment, and the states move deeper and deeper again. The LA converges once all the objects are in the innermost states (1 or 2, or 14 or 13), which we can observe from the end of the lines in the plot. It is important to note that, in this example, we have utilized a state depth of 12 in the EOMA. With a shallower state space, we could have followed the channels more instantaneously.

6.2 Results for Power Allocation

The simulation of the greedy solution and the channel-coefficient based solution to power allocation was done based on the groups established in the LA solution. For demonstrating the results of our approaches, we compared our solutions with those obtained by an exhaustive grid search. The exhaustive grid search was implemented in a step size of 0.001 ($\gamma = 0.001$). It was carried out for the same groups and the same values obtained from h through channel sounding as for the greedy and the channel-coefficient based solutions. We tested both the cases of doing power allocation based on instantaneous values, and on a time average ($\Delta t = 5$). In Fig. 7, we depict the results of the three approaches, for an average of $\Delta_t = 5$ samples of h.

As illustrated, the results of the greedy solution coincide with that of the exhaustive search, which means that giving more power to the user with strong channel coefficient, indeed, optimizes the sum rate of the system in the current configuration. The results of the channel-coefficient based solution have a better fairness among the users, where this is at a cost of attaining to a sub-optimal solution in terms of sum rate. The computations required for the greedy or the channel-coefficient based solution, depend on the number of users in each group, where $2L_c$ computations are needed for each group. By way of comparison, for L_c users in each group, we need to test $\left(\frac{P_n}{\gamma}\right)^{L_c}$ combinations for an exhaustive search.

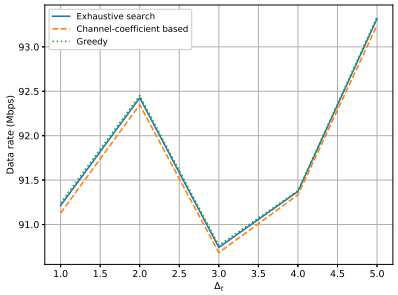


Fig. 7 Data rate for exhaustive, greedy and channel-based solution for groups of three users. Based on averages over 5 samples of $|h(t)|$.

Because the LA-based adaptive grouping solution accomplishes the partitioning of the users in favor of NOMA technology (i.e., users with distinct channel coefficients are grouped together), once the group is formulated, the objective of the power allocation may be changed to any other interesting form for NOMA, and thus different solutions can be further developed. In other words, the objective function of power allocation is not constrained to be the one requiring "sum-rate maximization".

7 Conclusions

In this paper, we have proposed a novel solution to the user grouping and power allocation problems in NOMA systems, also taking into consideration the stochastic nature of the users' channel coefficients. The grouping has been achieved by using the tabula rasa RL technique specified by the EOMA, and the simulation results presented show that a 100% accuracy for finding clusters of similar $h(t)$ over time, can be obtained in a limited number of iterations. In addition, our solution is able to follow the changes of $h(t)$, which makes our solution for grouping adaptive to changes in users' channel conditions, and the corresponding changes for their group associations. For power allocation, we proposed two solutions for the sum rate maximization with a QoS constraint for users. Our two-step solution offers flexibility with regard to both the grouping and power allocation phases, and can be used as stand-alone components of a NOMA system.

Acknowledgements We are very grateful to the anonymous Referees of the previous version of the paper, who suggested various modifications and changes. Their suggestions have enhanced the quality of this present version.

References

1. Y. Liu, Z. Qin, M. Elkashlan, Z. Ding, A. Nallanathan, and L. Hanzo, "Nonorthogonal Multiple Access for 5G and Beyond," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2347–2381, Dec. 2017.
2. M. Pischella and D. Le Ruyet, "NOMA-Relevant Clustering and Resource Allocation for Proportional Fair Uplink Communications," *IEEE Wireless Commun. Lett.*, vol. 8, no. 3, pp. 873–876, Jun. 2019.
3. J. Cui, Z. Ding, P. Fan, and N. Al-Dhahir, "Unsupervised Machine Learning-Based User Clustering in Millimeter-Wave-NOMA Systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 11, pp. 7425–7440, Nov. 2018.
4. S. Glimsdal and O. Granmo, "A Novel Bayesian Network Based Scheme for Finding the Optimal Solution to Stochastic Online Equi-Partitioning Problems," in *13th International Conference on Machine Learning and Applications*, Dec. 2014, pp. 594–599.
5. W. Gale, S. Das, and C. T. Yu, "Improvements to an Algorithm for Equipartitioning," *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 706–710, May 1990.
6. A. Shirvani and B. J. Oommen, "On Invoking Transitivity to Enhance the Pursuit-Oriented Object Migration Automata," *IEEE Access*, vol. 6, pp. 21 668–21 681, 2018.
7. ——, "On Utilizing the Pursuit Paradigm to Enhance the Deadlock-Preventing Object Migration Automaton," in *International Conference on New Trends in Computing Sciences (ICTCS)*, Oct. 2017, pp. 295–302.
8. M. A. Sedaghat and R. R. Müller, "On User Pairing in Uplink NOMA," *IEEE Trans. Wireless Commun.*, vol. 17, no. 5, pp. 3474–3486, May 2018.
9. S. Timotheou and I. Krikidis, "Fairness for Non-Orthogonal Multiple Access in 5g Systems," *IEEE Signal Process. Lett.*, vol. 22, no. 10, pp. 1647–1651, 2015.
10. Y. Liu, Z. Qin, M. Elkashlan, Y. Gao, and L. Hanzo, "Enhancing the Physical Layer Security of Non-Orthogonal Multiple Access in Large-Scale Networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1656–1672, 2017.
11. Z. Ding, Z. Yang, P. Fan, and H. V. Poor, "On the Performance of Non-Orthogonal Multiple Access in 5G Systems with Randomly Deployed Users," *IEEE Signal Process. Lett.*, vol. 21, no. 12, pp. 1501–1505, 2014.
12. N. Zhang, J. Wang, G. Kang, and Y. Liu, "Uplink Nonorthogonal Multiple Access in 5G Systems," *IEEE Commun. Lett.*, vol. 20, no. 3, pp. 458–461, 2016.
13. Y. Liu, Z. Qin, M. Elkashlan, A. Nallanathan, and J. A. McCann, "Non-Orthogonal Multiple Access in Large-Scale Heterogeneous Networks," *IEEE J. Sel. Areas Commun."*, vol. 35, no. 12, pp. 2667–2680, 2017.
14. Z. Ding, P. Fan, and H. V. Poor, "Random Beamforming in Millimeter-Wave NOMA Networks," *IEEE Access*, vol. 5, pp. 7667–7681, 2017.
15. J. Cui, Y. Liu, Z. Ding, P. Fan, and A. Nallanathan, "Optimal User Scheduling and Power Allocation for Millimeter Wave NOMA Systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1502–1517, 2017.
16. L. Zhu, J. Zhang, Z. Xiao, X. Cao, D. O. Wu, and X. Xia, "Joint Power Allocation and Beamforming for Non-Orthogonal Multiple Access (NOMA) in 5G Millimeter Wave Communications," *IEEE Trans. Wireless Commun.*, vol. 17, no. 9, pp. 6177–6189, Sep. 2018.
17. Y. Liu, Z. Ding, M. Elkashlan, and H. V. Poor, "Cooperative Non-Orthogonal Multiple Access with Simultaneous Wireless Information and Power Transfer," *IEEE J. Sel. Areas Commun."*, vol. 34, no. 4, pp. 938–953, 2016.
18. Y. Yin, Y. Peng, M. Liu, J. Yang, and G. Gui, "Dynamic User Grouping Based NOMA Over Rayleigh Fading Channels," *IEEE Access*, vol. 7, pp. 110 964–110 971, 2019.
19. X. Chen, A. Benjebbour, A. Li, and A. Harada, "Multi-User Proportional Fair Scheduling for Uplink Non-Orthogonal Multiple Access (NOMA)," in *IEEE 79th Vehicular Technology Conference (VTC Spring)*, May 2014, pp. 1–5.
20. F. Liu, P. Mähönen, and M. Petrova, "Proportional Fairness-Based User Pairing and Power Allocation for Non-Orthogonal Multiple Access," in *IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Aug. 2015, pp. 1127–1131.

21. M. L. Tsetlin, "Finite Automata and Modeling the Simplest Forms of Behavior," in *Mathematics in Science and Engineering*, 1973, vol. 102, pp. 3–83.
22. S. Lakshmivarahan, *Learning Algorithms: Theory and Applications*. New York: Springer, 1981.
23. K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Courier Corporation, May 2013.
24. S. Lakshmivarahan and M. A. L. Thathachar, "Absolutely Expedient Algorithms for Stochastic Automata," *IEEE Trans. Syst. Man Cybern.*, vol. 3, pp. 281–286, 1973.
25. B. J. Oommen and M. Agache, "Continuous and Discretized Pursuit Learning Schemes: Various Algorithms and Their Comparison," *IEEE Trans. Syst. Man Cybern.: B Cybern.*, vol. 31, no. 3, pp. 277–287, 2001.
26. X. Zhang, O.-C. Granmo, and B. J. Oommen, "Discretized Bayesian Pursuit - A New Scheme for Reinforcement Learning," in *Proceedings of IEA-AIE 2012*, Dalian, China, Jun. 2012, pp. 784–793.
27. A. S. Poznyak and K. Najim, *Learning Automata and Stochastic Optimization*. Springer, 1997, vol. 3.
28. A. Yazidi, X. Zhang, L. Jiao, and B. J. Oommen, "The Hierarchical Continuous Pursuit Learning Automation: A Novel Scheme for Environments With Large Numbers of Actions," *IEEE Trans. Neural. Netw. Learn. Syst.*, vol. 31, no. 2, pp. 512–526, 2020.
29. X. Zhang, L. Jiao, B. J. Oommen, and O. Granmo, "A Conclusive Analysis of the Finite-Time Behavior of the Discretized Pursuit Learning Automaton," *IEEE Trans. Neural. Netw. Learn. Syst.*, vol. 31, no. 1, pp. 284–294, 2020.
30. A. Shirvani, "Novel Solutions and Applications of the Object Partitioning Problem," Ph.D. dissertation, Carleton University, 2018.
31. M. Pätzold, *Mobile Radio Channels*, 2nd ed. Chichester: Wiley, 2012.
32. H. Xing, Y. Liu, A. Nallanathan, Z. Ding, and H. V. Poor, "Optimal Throughput Fairness Tradeoffs for Downlink Non-Orthogonal Multiple Access Over Fading Channels," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 3556–3571, Jun. 2018.
33. J. Kang and I. Kim, "Optimal User Grouping for Downlink NOMA," *IEEE Wireless Commun. Lett.*, vol. 7, no. 5, pp. 724–727, Oct. 2018.
34. L. Zhu, J. Zhang, Z. Xiao, X. Cao, and D. O. Wu, "Optimal User Pairing for Downlink Non-orthogonal Multiple Access (NOMA)," *IEEE Wireless Commun. Lett.*, vol. 8, no. 2, pp. 328–331, Apr. 2019.
35. 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.
36. O. Granmo, B. J. Oommen, S. A. Myrer, and M. G. Olsen, "Learning Automata-Based Solutions to the Nonlinear Fractional Knapsack Problem With Applications to Optimal Resource Allocation," *IEEE Trans. Syst. Man Cybern.: B Cybern.*, vol. 37, no. 1, pp. 166–175, Feb. 2007.
37. B. J. Oommen, "Stochastic Searching on the Line and its Applications to Parameter Learning in Nonlinear Optimization," *IEEE Trans. Syst. Man Cybern.: B Cybern.*, vol. 27, no. 4, pp. 733–739, 1997.
38. M. Stege, J. Jelitto, N. Lohse, M. Bronzel, and G. Fettweis, "A Stochastic Vector Channel Model-Implementation and Verification," in *IEEE 50th Vehicular Technology Conference*, vol. 1, Sep. 1999, pp. 97–101 vol.1.
39. W. Gale, S. Das, and C. T. Yu, "Improvements to an Algorithm for Equipartitioning," *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 706–710, May 1990.
40. Y. Liu, M. Elkashlan, Z. Ding, and G. K. Karagiannidis, "Fairness of User Clustering in MIMO Non-Orthogonal Multiple Access Systems," *IEEE Commun. Lett.*, vol. 20, no. 7, pp. 1465–1468, Jul. 2016.