# 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 $\rho_1$ 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, $\Delta^*$, and the solution algorithm determines a partitioning, say $\Delta^+$. The solution is optimal if $\Delta^+ = \Delta^*$. The initialization of the objects before partitioning starts is indicated by $\Delta^0$.

### 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, $\Psi$. 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, $\tau$. 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, $\Psi$, 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, $\Psi_Q$. 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, $\Psi$, includes both the queries that "survive" the pursuit filtering, and the artificially generated queries. Again, $\Psi_Q$ indicates the number of queries made by the Query Generator. Besides, we introduce the parameter $\Psi_T$ 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 $\Delta^*$. To achieve this, we introduce the parameter $\gamma$, 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 $\eta_k$ parameter, where $k \in \{1,...,K\}$, is the number of objects in each partition. To determine $\gamma$, we need to check all possible query pairs, observe if the objects in a query are grouped both in $\Delta^+$ and $\Delta^*$, 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 $\Lambda$, 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 $\Lambda$, 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 $\Lambda$, 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 $\rho_1$, $\iota_0 = 0$ 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 $\rho_1$ 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 $\rho_1$, $\rho_2$ and $\rho_3$ respectively.

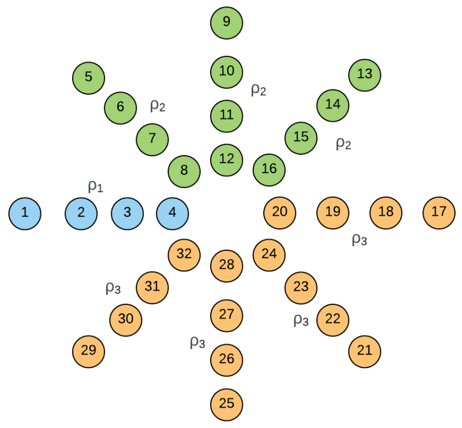

**Fig. 1.** Example of partition links in GCD with 3 partitions and 4 states as described in the text.

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 $\Lambda$ 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 $\Upsilon$, where each entry $Q = \{o_i, o_j\}$.
- Initialized $\theta_i$ for all objects. Initially all $\theta_i$, where $i \in \{1, 2, ..., O\}$, is given a random boundary state, where we have $\Lambda$ 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 $\Lambda$ 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.
- $\theta_i$ 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 $\rho_k$, 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 $\Upsilon$
3:     if $\theta_i$ 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 $o_j$.
1: if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S \neq 0$ then        // Neither are in boundary
2:     $\theta_i = \theta_i + 1$
3:     $\theta_j = \theta_j + 1$
4: else if $\theta_i \bmod S \neq 0$ and $\theta_j \bmod S = 0$ then      // $o_j$ is in boundary
5:     $\theta_i = \theta_i + 1$
6:     $temp = \theta_j$                                          // Store the state of $o_j$
7:     $o_l$ = unaccessed object in group of staying object ($o_i$) closest to boundary
8:     $\theta_j = \theta_i$
9:     $\theta_l = temp$
10: else if $\theta_i \bmod S = 0$ and $\theta_j \bmod S \neq 0$ then     // $o_i$ is in boundary
11:     $\theta_j = \theta_j + 1$
12:     $temp = \theta_i$                                         // Store the state of $o_i$
13:     $o_l$ = unaccessed object in group of staying object ($o_j$) closest to boundary
14:     $\theta_i = \theta_j$
15:     $\theta_l = temp$
16: else                                                      // Both are in boundary states
17:     $temp = \theta_i$ or $\theta_j$                  // Store the state of moving object, $o_i$ or $o_j$
18:     $\theta_i = \theta_j$ or $\theta_j = \theta_i$       // Put moving object and staying object together
19:     $o_l$ = unaccessed object in group of staying object closest to boundary
20:     $\theta_l = temp$                        // Move $o_l$ to the old state of moving object
21: 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 $\Delta^*$ 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, $\Pi_{o_i,o_j}$ is the probability of $o_i$ and $o_j$ being accessed together and being together in $\Delta^*$. 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 $\frac{Q}{K} = 10$. Table 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 $\Psi$ indicates the number of queries considered by the LA, $\Psi_Q$ the total number of queries generated, and $\Psi_T$ the queries made from the concept of transitivity in the TPEOMA. For PEOMA, we have to include the parameter $\kappa$, 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 $\tau$, indicating the threshold for whether a query should be considered or not [8, 10].

| Type | Noise | $\gamma$ | $\Delta^+ = \Delta^*$ | Not Conv. | $\Psi$ | $\Psi_Q$ | $\Psi_T$ | $\kappa$ | $\tau$ |
|---|---|---|---|---|---|---|---|---|---|
| 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 | $\gamma$ | $\Delta^+ = \Delta^*$ | Not Conv. | $\Psi$ | $\Psi_Q$ | $\Psi_T$ | $\kappa$ | $\tau$ |
|---|---|---|---|---|---|---|---|---|---|
| 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)