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

Rebekka Olsson Omslandseter, Lei Jiao, B. John John Oommen

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].

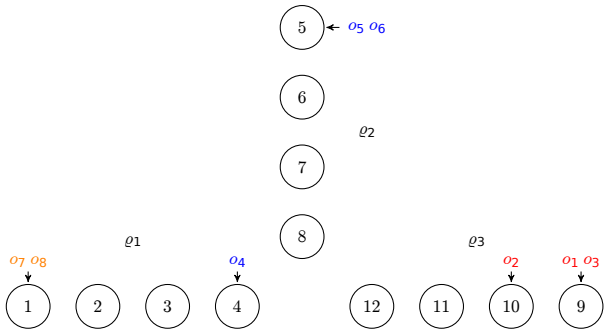

Fig. 1. Example of objects stuck in a Standstill Situation.

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

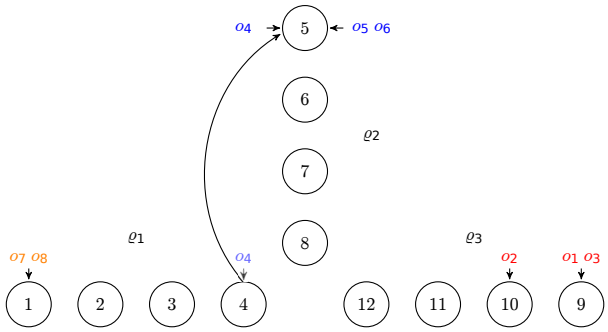

Fig. 2. Example of the Penalty functionality for the Standstill Situation.

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