DETERMINISTIC LEARNING AUTOMATA SOLUTIONS TO THE OBJECT PARTITIONING PROBLEM

B. John Oommen
D.C.Y. Ma

SCS-TR-84
November 1985

School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

*This research was supported by the Natural Science and Engineering Research Council of Canada.

Deterministic Learning Automata Solutions To The Object Partitioning Problem[+]

B. J. Oommen and D. C. Y. Ma
School of Computer Science
Carleton University
Ottawa, K1S 5B6
CANADA.

ABSTRACT

Let $\mathcal{A} = \{A_1, \ldots, A_W\}$ be a set of W objects to be equi-partitioned into R classes $\{P_1, \ldots, P_R\}$. The objects are accessed in groups, their joint access probabilities being unknown. The intention is that the objects accessed more frequently together are located in the same class. This problem has been known to be NP-hard [30]. In this paper, we propose three deterministic learning automata solutions to the problem. Although the first two are ε-optimal, they seem to be practically feasible only when W is small. The last solution, which uses a new learning automaton, demonstrates an excellent partitioning capability. Experimentally, this solution converges an order of magnitude faster than the best known algorithm in the literature [30].

+ Partially Supported by the Natural Sciences and Engineering Research Council of Canada.

I. INTRODUCTION

Let $\mathcal{A} = \{A_1, ..., A_W\}$ be a set of W objects. We intend to partition $\mathcal{A}$ into R classes $\{P_1, ..., P_R\}$ where each class has W/R objects. Further, we intend to partition them in such a way that the objects that are accessed (used) more frequently together lie in the same class. This problem is called the Object Partitioning Problem (OPP). Indeed, to render the problem non-trivial, we assume that these joint access probabilities are unknown.

There are various applications for the Object Partitioning Problem (OPP). In a library environment [23], the set $\mathcal{A}$ could represent the set of disciplines (e.g. history, geography, biology, etc.) and the partitioning would represent the physical location of the books and documents in these individual disciplines. Thus, assuming that a person working in electronics required books in mathematics more frequently than books in geography, it is more desirable that books in the former two disciplines (i.e., electronics and mathematics) be located in closer proximity than books in the latter two disciplines (i.e., electronics and geography). We shall present below some other computer science applications of the OPP :

a) Let $\mathcal{A}$ be a set of attributes belonging to a relation in the normal form. Because of the size of $\mathcal{A}$, it may be that these attributes cannot all be physically incorporated in one single relation. We would then like to partition the attributes into sub-relations in such a way that the attributes accessed simultaneously in a query are located in the same sub-relation [4,5,29,30].

b) Let $\mathcal{A}$ be a set of records, and it is required to partition these records so that the jointly accessed records are available on the same block (or track of the disk). The solution to the OPP can directly be used to solve this problem. So too, as stated in [30], segment clustering in IMS trees, can be considered as a record clustering problem [24].

c) In a distributed data base, the files that are distributed can be located in the various sites. In this case, it is desirable that files that are jointly accessed are physically located in the same geographical site. Obviously this will minimize communication costs [12,29].

Various other applications of the OPP such as the selection of dependent terms and the automatic indexing problem are found in the literature [11,27,28]. Our aim is to obtain a general solution to the OPP which, with minor modifications, lends itself as a solution to any of the above mentioned problems.

The most obvious method to solve this problem is to process the queries and to maintain a frequency count of the accesses for various combinations of the objects. Since the number of combinations is usually very high, we discourage the use of such statistics, on the counts of both time and storage requirements.

A closely related problem that has been extensively studied is the one of organizing a linked list with the aim of having the records organized in the descending order of their access probabilities $\{s_i\}$ [1-3,6,7,13,19-22]. Of course, in this case too, one can envision a strategy of "learning" the positions of the individual records by actual keeping a count on the number of times they are accessed (i.e. estimating their access probabilities). However, this requires excess memory and this motivated the search for self-organizing algorithms. Due to the similarity of our solution to the OPP and various list organizing strategies, we shall briefly describe some algorithms for the latter problem.

McCabe [8, pp.398-399; 13] was the first to propose a solution to this problem. His solution rendered the list dynamically self-organizing and it involved moving an element to the front of the list every time it was accessed. Many other researchers [3,6,7,21,22] have also extensively studied the Move-To-Front rule and various properties of its limiting convergence characteristics are available in the literature.

McCabe [8, pp.398-399; 13] also introduced a scheme which is called the transposition rule. This rule requires that an accessed element is interchanged with its preceding element in the list, unless, of course, it is at the front of the list. Much literature is available on the transposition rule [1-3,8,13,21] but particularly important is the work of Rivest [21] and Tenenbaum *et al* [1] who extensively studied this rule and suggested its generalizations -- the Move-k-Ahead rule and the POS(k) rule. In the former, the accessed element is moved k positions forward to the front of the list unless it is found in the first k positions - in which case it is moved to the front. The POS(k) rule moves the accessed element to position k of the list if it is in positions k+1 through n. It transposes it with the preceding element if it is in positions 2 through k. If it is the first element it is left unchanged.

Rivest [21] showed that the limiting behaviour of the transposition rule (quantified in terms of the average number of probes) was never worse than that of the Move-To-Front rule. He conjectured that the transposition rule has lower expected cost than any other reorganization scheme. He also conjectured that the move-k-ahead rule was superior to the move-k+1-ahead rule; but as yet this is unproven. Tenenbaum and Nemes proved various results for the POS(k) rule primarily involving a particular degenerate distribution which is easy to analyze. Their results seem to strengthen Rivest's conjecture.

Recently, Oommen and Hansen [20] proposed two learning algorithms in which the elements of the list adaptively learn to find their place. In [20], the first algorithm is essentially a move-to-the-front algorithm with the exception that on the n^{th} access, the accessed element is moved to the front of the list with a probability f(n). This probability is systematically decreased every time an element is accessed in such a way that the scheme is both absorbing and expedient, i.e., if $s_i > s_j$, then the probability of absorption into an arrangement in which R_i precedes R_j (R_i, $R_j \in \{R_1, ..., R_N\}$, the given set of elements) is always greater than 0.5.

The second algorithm in [20] is far more powerful. This algorithm is a move-to-rear scheme in which the accessed element R_j is moved to the **rear** of the list with a probability q_j. This quantity q_j is progressively decremented every time **the element** is accessed. Thus, ultimately there is no move operation performed on the list. Hence, this scheme too is absorbing in its Markovian representation and so could converge into any one of its N! arrangements. However, in this case, it has been shown that the probability of converging to the optimal arrangement can be made as close to unity as desired. The algorithms in [20] are the only two known absorbing algorithms discussed in the literature.

Considering the body of work done in the latter problem, one hopes to obtain an adaptive solution to the problem currently studied, namely the OPP. Further, we would like the solution to avoid estimation of the access probabilities of the combinations of the objects. Unfortunately, the solutions for the list organizing problem are not directly usable because the assumption that the objects are accessed independently is quite meaningless in this context. Further the question of physically moving the objects after every query (or access) seems quite unreasonable, especially in the case of attribute partitioning or distributed file allocation. Thus, the need for a substantially different scheme is obvious.

The best known adaptive solution available for the OPP is the one due to Yu *et al* [30]. Its closest competitor is possibly the technique due to Hammer *et al* [4]. The latter is a clever algorithm, which can be thought of as a "hill-climbing" technique that tries to converge to the optimal partitioning by using a heuristic approach. At its time it probably was the best algorithm to solve the attribute partitioning problem. In [30], Yu *et al* proposed an adaptive scheme which was truly ingenuous. We shall describe this scheme in a later section, but we shall comment in passing that the authors of [30] demonstrate the superiority of their algorithm to the one in [4]. We refer the reader to [30] for a good review and comparison of the various algorithms.

In this paper we shall present some deterministic learning automata solutions to the OPP. Learning automata have been used in the literature to model biological learning systems and also to learn the optimal action which an environment offers. The learning is achieved by actually interacting with the environment and processing its responses to the actions that are chosen. Such automata have various applications such as parameter optimization, statistical decision making and telephone routing [10,14-16]. Since the literature on learning automata is extensive, we refer the reader to a paper by Narendra and Thathachar [14] and a book by Lakshmivarahan [10] for a review of the families and applications of learning automata.

The learning process of an automaton can be described as follows: The automaton is offered a set of actions by the environment with which it interacts, and it is constrained to choose one of these actions. When an action is chosen, the automaton is either rewarded or penalized by the environment with a certain probability. A learning automaton is one which learns the optimal action, i.e. the action which has the minimum penalty probability, and eventually chooses this action more frequently than other actions.

Stochastic learning automata can be classified into two main classes : (a) Fixed Structure Stochastic Automata (FSSA) and (b) automata whose structure evolve with time. Some examples of the former type are the Tsetlin, Krinsky and Krylov automata [25,26]. Although the latter automata are called variable structure stochastic automata, because their transition and output matrices are time varying, in practice, they are merely defined in terms of action probability updating rules which are either of a continuous [9,10,14-16] or discrete nature [17,18]. Although throughout this paper we will only be considering Fixed Structure Stochastic Automata (FSSA), automata with a variable structure are generally much faster in their convergence. We are currently investigating the use of automata of the latter class to solve the OPP.

For the rest of this section, in all brevity, we discuss the fundamentals of learning automata. In the next section we present the algorithm due to Yu *et al* [30] which, in our judgement, is the best known adaptive scheme. We shall then present our solutions to the OPP and discuss the experimental results we have obtained.

I.1 Fundamentals

A FSSA is a quintuple ($\boldsymbol{\alpha}$, $\boldsymbol{\Phi}$, $\boldsymbol{\beta}$, F, G) where :

(i) $\boldsymbol{\alpha} = \{\alpha_1, ..., \alpha_R\}$ is the set of actions that it must choose from.

(ii) $\boldsymbol{\Phi}$ is its set of states.

(iii) $\boldsymbol{\beta} = \{0, 1\}$ is its set of inputs. The input '1' represents a penalty.

(iv) F is a map from $\boldsymbol{\Phi} \times \boldsymbol{\beta}$ to $\boldsymbol{\Phi}$. It defines the transition of the internal state of the automaton on receiving an input. F may be stochastic.

(v) G is a map from $\boldsymbol{\Phi}$ to $\boldsymbol{\alpha}$, and determines the action taken by the automaton if it is in a given state. With no loss of generality G is deterministic [14].

The selected action serves as the input to the environment which gives out a response $\beta(n)$ at time 'n'. $\beta(n)$ is an element of $\boldsymbol{\beta} = \{0,1\}$ and is the fedback response of the environment to the automaton. The environment penalizes (i.e., $\beta(n) = 1$) the automaton with the penalty c_i, where,

$$c_i = \Pr[\beta(n) = 1 \mid \alpha(n) = \alpha_i] \qquad (i = 1 \text{ to } R).$$

Thus the environment characteristics are specified by the set of penalty probabilities $\{c_i\}$ ($i = 1$ to R). On the basis of the response $\beta(n)$ the state of the automaton $\phi(n)$ is updated and a new action chosen at (n+1).

The $\{c_i\}$ are unknown initially and it is desired that as a result of interaction with the environment the automaton arrives at the action which evokes the minimum penalty response in an expected sense. It may be noted that if c_L is the minimum penalty probability, then, if we define,

$$P_i(n) = \Pr[\alpha(n) = \alpha_i],$$

$$P_L(n) = 1, \quad P_i(n) = 0 \quad \text{for } i \neq L$$

achieves this result. Automata are designed with this solution in view.

I.2 Learning Criteria

With no apriori information, the automaton chooses the actions with equal probability. The expected penalty is thus initially M_0, the mean of the penalty probabilities.

An automaton is said to learn **expediently** if, as time tends towards infinity, the expected penalty is less than M_0. We denote the expected penalty at time 'n' as $E[M(n)]$. The automaton is said to be **optimal** if $E[M(n)]$ equals the minimum penalty probability in the limit as time goes towards infinity.

It is **ε-optimal** if in the limit $E[M(n)] < c_L + \varepsilon$, where c_L is the minimum penalty probability, for any arbitrary $\varepsilon > 0$. This could be achieved by a suitable choice of some parameter of the automaton, for example, the number of states of the FSSA. In this case, the limiting value of E[M(n)] can be as close to c_L as desired.

II. THE BASIC ADAPTIVE METHOD

Yu *et al* [30] proposed an algorithm to solve the OPP, which they have called the Basic Adaptive Method (BAM). In our reckoning, we believe that this is the best known algorithm for the OPP, and its advantages over the technique due to Hammer *et al* [14] are well defended in [30]. We describe below the BAM in the set up when exactly two objects are accessed per query. (The case when more than two objects are accessed has also been handled in [30]).

The three major advantages of the BAM are (i) the method is adaptive, (ii) it does not involve computing any query statistics and (iii) as opposed to list organization, the objects are not moved on processing every query. The strategy is as follows : Associated with the set of physical objects $\mathcal{A} = \{A_1, A_2, ..., A_W\}$ is a set of **abstract** objects $\{O_1, O_2, ..., O_W\}$. The BAM manipulates **these abstract** objects as the queries are processed. Notice that this can be done in the background, in a way that is invisible to the user. Periodically, the set of physical objects $\mathcal{A}$ are re-partitioned so to conform to the partitioning dictated by the BAM. This periodical repartitioning of $\mathcal{A}$ can be done when queries on the actual physical objects are not expected -- for example at back-up times. Initially the BAM assigns for each abstract object O_i a real number X_i. On processing a query requesting the physical objects A_i and A_j, the quantities X_i and X_j are moved toward their centroid by a small constant Δ_1. This tends to bring them into the same cluster. Subsequently, two distinct random indices p and q are chosen, $1 \leq p,q \leq W$, and X_p and X_q are drawn away from **their** centroid by a constant Δ_2. The techniques of "determining" Δ_1, Δ_2 and the initial X_i's are informally discussed in [30]. After a fairly long time (T accesses), the partitions are created based on the proximity of the X_i's. The technique for creating these partitions is also discussed in [30]. For the sake of completeness, we present an algorithmic version of the BAM.

Algorithm BAM

Input : The abstract objects $\{O_1, ..., O_W\}$, two parameters Δ_1 and Δ_2, and the stream of queries. Every query is conceptually a pair (A_i, A_j).

```
Output :  A periodic clustering of the objects into R partitions.

Method :  Initialize arbitrarily $X_i$ for all $1 \le i \le W$, where $-\infty < X_i < \infty$.
          Repeat
              For a sequence of T queries do
                  ReadQuery ($A_i$, $A_j$)
                  If ($X_i < X_j$) then
                      $X_i = X_i + \Delta_1$
                      $X_j = X_j - \Delta_1$
                  Else
                      $X_i = X_i - \Delta_1$
                      $X_j = X_j + \Delta_1$
                  EndIf
                  Select Randomly distinct indices p, q, where $1 \le p, q \le W$
                  If ($X_p < X_q$) Then
                      $X_p = X_p - \Delta_2$
                      $X_q = X_q + \Delta_2$
                  Else
                      $X_p = X_p + \Delta_2$
                      $X_q = X_q - \Delta_2$
                  EndIf
              EndFor
              Print out Partitions based on proximity of $\{X_i\}$
          ForEver
End Algorithm BAM
```

We shall now proceed to discuss the FSSA solutions to the OPP.

III. SOLUTION OF THE OPP USING TSETLIN & KRINSKY AUTOMATA

The first two solutions for the OPP which we propose involve the Tsetlin and the Krinsky learning automata.

Tsetlin [25,26], who initiated the work on FSSA was the first to design a machine which actually had expedient (and even ε-optimal) learning behaviour. It was initially called the Linear Tactic. The K-action Tsetlin automaton, $L_{KN,K}$, has

KN states $\{\phi_1, \phi_2, ..., \phi_{KN}\}$ and chooses one of the K actions $\{\alpha_1, \alpha_2, ..., \alpha_k\}$. N is called the "Depth" of the automaton. Explicitly, $L_{KN,K}$ is defined as follows :

$$L_{KN,K} = \left(\{\phi_1, \phi_2, ..., \phi_{KN}\}, \{\alpha_1, \alpha_2, ..., \alpha_k\}, \{0, 1\}, L(\cdot, \cdot), G(\cdot) \right)$$

The G(·) map is deterministic and is defined as :

$$G(\phi_j) = \alpha_i \qquad \text{if} \quad (i-1)N + 1 \le j \le iN \qquad (1)$$

Observe that since this means that the automaton chooses α_1 if it is in any of the first N states, it chooses α_2 if it is in any of the states from ϕ_{N+1} to ϕ_{2N} , etc, the states of the automaton are automatically partitioned into groups, each group representing an action.

The L(· , ·) map is **deterministic** and is described as below :

(1) If $\beta = 0$ (it gets a favourable response), the L map requires that the automaton go towards the most extreme state corresponding to that action -- $\phi_{(j-1)N+1}$ for α_j -- one step at a time.

(2) If $\beta = 1$ (it gets an unfavourable response), the automaton moves towards the boundary state -- ϕ_{jN} if α_j is the action chosen -- one step at a time. If it is in the boundary state for the action it moves to the boundary state for the next action. Thus, for all $1 \le j \le k$.

$$\begin{aligned} \text{Thus,} \quad L(\phi_i, 0) &= \phi_{i-1} && \text{if} \quad (j-1)N + 1 < i \le jN \\ &= \phi_i && \text{if} \quad i = (j-1)N + 1 \qquad (2) \\ \text{And,} \quad L(\phi_i, 1) &= \phi_{i+1} && \text{if} \quad (j-1)N + 1 \le i < jN \\ &= \phi_{((j+1)N) \bmod KN} && \text{if} \quad i = jN \qquad (3) \end{aligned}$$

The map is drawn in Figure I for the case when K = 2. We now state the following theorem which was proved two decades ago [25].

Theorem I

The $L_{KN,K}$ automaton is ε-optimal in all random environments in which the minimum penalty probability is less than 0.5. • • •

Following the introduction of the Tsetlin automaton various scientists proposed other FSSA which had good learning characteristics. One of these automata, $Z_{KN,K}$, the Krinsky automaton, is defined possessing KN states $\{\phi_1, ..., \phi_{KN}\}$ and K actions $\{\alpha_1, ..., \alpha_k\}$ as follows :

$$Z_{KN,K} = \left(\{\phi_1, ..., \phi_{KN}\}, \{\alpha_1, ..., \alpha_k\}, \{0, 1\}, Z(\cdot, \cdot), G(\cdot) \right)$$

The G map is deterministic and is defined as in (1), exactly as the G map of the Tsetlin automaton. Thus the automaton chooses α_i if it is in any state ϕ_j, where $(i-1)N+1 \le j \le iN$.

The Z map is deterministic. On penalized, the automaton behaves just like the Tsetlin automaton. Thus,

$$Z(\phi_i, 1) = L(\phi_i, 1) \qquad 1 \le i \le KN, \qquad (4)$$

where $L(\phi_i, 1)$ is defined by (3) above. However, on being rewarded, the automaton goes directly to the most extreme state corresponding to that action -- $\phi_{(j-1)N+1}$ for action α_j. Explicitly,

$$Z(\phi_i, 0) = \phi_{(j-1)N+1} \qquad \text{if } (j-1)N+1 \le i \le jN \qquad (5)$$

The transition map of the Krinsky automaton is shown in Figure II for the case when $K = 2$. As opposed to the Tsetlin automaton, the Krinsky automaton has the following property [26] :

Theorem II

The Krinsky automaton is ε-optimal in **all** random environments. • • •

The solution of the OPP using these automata is now straightforward. The actions of the automaton are the various partitions of the abstract objects, and each action has N states associated with it. Assume that the automaton chooses action α_k at the time instant 'n'. At the (n+1)st time instant, the query (A_i, A_j) either supports the partitioning represented by α_k or contradicts it. Correspondingly, the learning automaton is rewarded or penalized respectively, and the learning process continues by having the automaton interact with the "query system" -- which merely serves the purpose of the environment.

The practical implementation of such a learning machine exactly entails the maintenance of **two integer** memory locations. The first of these two stores the action chosen by the machine and thus can take a value ranging from 1 to K. The second, which can take a value from 1 to N, represents the internal state corresponding to that action (i.e. how deep it is in its "assurance" of its choice involving the actions). Apart from these, a **static** memory array must be allocated to "remember" the partitioning represented by each action. If there are W objects and R classes, this partitioning is most conveniently coded as a W-digit base-R integer. Thus, if $W = 6$ and $R = 3$, the number 001122 represents the partition $\{(A_1,A_2), (A_3,A_4), (A_5,A_6)\}$. Observe that this static array can be computed, and further, need be computed exactly **once**.

III.1. Experimental Results and Comments

The OPP was simulated for two sets of objects with various probabilities of being jointly accessed. The Tsetlin automaton, the Krinsky automaton and the BAM were each used to perform the learning of the optimal partitioning. In each case ten experiments were conducted, and in every experiment joint accesses on the objects were made till the learning process converged. In the first set, W, the number of objects, was four and R, the number of groups, was two. In the second set, W and R were six and two, respectively. In each case, queries were randomly generated based on an underlying distribution unknown to the partitioning algorithm. If π_i was the partition in which A_i was **to be** located, then, these queries obeyed the following distribution :

$$\sum_{A_j \in \pi_i} \Pr[\, A_i, A_j \text{ accessed together} \,] = p, \qquad (6)$$

Observe that if p = 1, then the partitioning is ideal, i.e., queries will involve only objects in the same partition. As the value of p decreases, the queries will be **decreasingly informative** about the solution of the OPP.

In the case of the BAM, the initial X_i's were randomly chosen in [0, 100], and Δ_1 and Δ_2 were set to have the same value, namely, unity. Observe that in this case a very tight initial distribution of X_i's could lead to the convergence involving a single class. The range [0, 100] was used as a standard in all our experiments because it caused the BAM to yield good partition results especially when W was large.

In the case of the Tsetlin and Krinsky automata, the number of states per action was 10. To render the comparison between the various algorithms meaningful, the FSSA algorithms were reckoned to have "converged" if they reached the most internal state of a particular action. Observe that this is quite relevant, for when this occurs, the probability of emerging from that action is very low. Further, this is quite comparable to the notion of the convergence of the BAM discussed in [30].

Table I summarizes the results that have been obtained. From the table we observe that the automata solution to the OPP are far superior to the BAM. For example, in the case when W = 6 and R = 2, when p = 0.8, the BAM on the average takes 225 iterations to first **reach** the optimal solution, and finally after 240 iterations, it converges to this solution. In this case the figures for the Tsetlin automaton are 10 and 25 iterations, respectively. Similarly, the figures for the Krinsky automaton are 16 and 17 iterations, respectively. Notice that in this case, if we consider convergence as a criterion, the Tsetlin automaton is 9.6 times faster than the BAM, and the Krinsky automaton is 14.1 times faster than the BAM. Such results are **typical**.

No. Objects	No. Classes	p	# iterations BAM	# iterations Tsetlin	# iterations Krinsky
4	2	0.6	(80, 135)	(15, 30)	(23, 25)
		0.7	(75, 120)	(6, 20)	(20, 21)
		0.8	(60, 85)	(6, 17)	(7, 8)
		0.9	(50, 80)	(5, 14)	(6, 7)
6	2	0.6	(350, 375)	(75, 100)	(21, 22)
		0.7	(280, 295)	(20, 35)	(20, 21)
		0.8	(225, 240)	(10, 25)	(16, 17)
		0.9	(200, 240)	(10, 20)	(11, 12)

Table I : Comparison of the BAM and the Tsetlin and Krinsky Automata solutions for the OPP. In the latter cases, the depth of memory is 10.

Notation : p -- probability defined by (6).
Number of iterations is given as a pair (x,y), where x is the average number of iterations for the grouping to be obtained, and y is the average number of iterations for the algorithm to converge to the grouping.

Observe that although these automata do require KN states for K actions, neither the Tsetlin nor the Krinsky automaton does any computation requiring query statistics. This is essentially the advantage of learning automata over traditional adaptive schemes [14]. However, although these automata have such good learning properties, they are practically feasible only when W and R are small. This is because of the following theorem.

Theorem III

For any particular value of W and R, the number of actions that a Tsetlin (or Krinsky) automaton solving the OPP must possess is $\frac{W!}{(M!)^R R!}$, where M = W/R.

Proof :

The number of actions that the automaton possesses must be equal to the number of equal sized partitions that can be made from W objects into R classes.

Let this number be $f(W,R)$. Trivially, this implies that in each class there will be M objects, since $M = W/R$.

Suppose the object O_1 is in any one class. Then, the number of distinct ways by which this class can be filled is ${}_{W-1}C_{M-1}$, where ${}_{A}C_{B}$ is the binomial coefficient having the value $A! / (B!(A-B)!)$. This leaves us with R-1 classes to be filled and W-M objects with which to fill them. Thus, $f(W,R)$ obeys the difference equation :

$$f(W,R) = {}_{W-1}C_{M-1} \cdot f(W-R, R-1), \qquad \text{where } M = W/R \tag{7}$$

Substituting, it can be seen that $f(W,R) = W! / (M!)^R R!$ satisfies (7) and the theorem is proved. • • •

Using the above theorem it is easy to see that if the number of objects is 12 and the number of classes is 3, the automaton must possess 5775 actions. Clearly, this renders these automata impractical, unless there is a straightforward way to code the actions into their partition representations such that on processing a query a linear search will not have to be done. This is currently being investigated.

We shall now consider the next FSSA which is in general slower than the above automata, but which is still much faster than the BAM, and yet feasible when the number of objects to be partitioned is large.

IV. THE OBJECT MIGRATING AUTOMATON SOLUTION

We define the Object Migrating Automaton (OMA) as a quintuple in which there are **only** R actions, each action representing a certain **class**. Further for each action, there is a fixed number of states N. As opposed to the previous automata in which the **automata** can move from one action to another, in this case, we have **all** the W abstract objects moving around in the automaton. If the abstract object O_i is in action α_k, this is to signify that this abstract object will be in the class numbered k. Thus, If O_i and O_j are in the same class and the query requests (A_i,A_j), the OMA is rewarded. Otherwise, it is penalized. Observe that this means that if all the objects are in the same class, the OMA will obviously never be penalized, and so the automaton will tend to converge to such an arrangement. Clearly this is undesirable and so this scenario will have to be explicitly prohibited. We shall show below how this is done.

For each action α_k, there is a set of states $\{\phi_{k1}, \ldots, \phi_{kN}\}$, where N is the depth of memory and $1 \le k \le R$, the number of classes (partitions) desired. With no loss of generality, we assume ϕ_{k1} to be the most internal state of that action and ϕ_{kN} to be the boundary state. When a pair of physical objects (A_i, A_j) are accessed, if the **abstract** objects O_i and O_j are in the same class α_k, both of them are rewarded and are moved toward the most internal state, ϕ_{k1}, of that action, one step at a time. See Figure III(a). If, however, the abstract objects lie in different classes, say α_k and α_m respectively, (i.e. O_i is in state ξ_i, where $\xi_i \in \{\phi_{k1}, ..., \phi_{kN}\}$, and O_j is in state ξ_j, where $\xi_j \in \{ \phi_{m1}, ..., \phi_{mN}\}$), they are moved away from ϕ_{k1} and ϕ_{m1} as follows :

a) If $\xi_i \neq \phi_{kN}$ and $\xi_j \neq \phi_{mN}$, then move O_i and O_j one state toward ϕ_{kN} and ϕ_{mN}, respectively. See Figure III (b).

b) If exactly one is in the boundary state, (i.e. either $\xi_i = \phi_{kN}$ or $\xi_j = \phi_{mN}$, but not both), then move the object which is **not** in the boundary state towards its boundary state. See Figure III (c).

c) If both are in the boundary state, (i.e. $\xi_i = \phi_{kN}$ and $\xi_j = \phi_{mN}$), then one of them, say O_i, will be moved to the boundary state of α_m. In this case, since this will result in an excess of objects in α_m, one of the objects in α_m which is **not** accessed is moved to the boundary state of α_k. We choose to move the one closest to ξ_j. See Figure III(d).

As in the case of the BAM, periodically, i.e. after T accesses, the physical objects are partitioned as specified by the location of the abstract objects. Observe that in this case, no clustering based on the proximity of these objects is required as in the BAM, because the OMA itself specifies whether the objects are to be in the same class or not.

For the sake of completeness, we now algorithmically give the above described technique. For simplicity, as in the case of the other FSSA, we merely update the indices of the states in which the objects are .

Algorithm OMA

Input : The abstract objects $\{O_1, ..., O_W\}$ and the number of states N per action. Also coming into the algorithm is a stream of queries (A_i, A_j).

Output : A periodic clustering of the objects into R partitions.

Notation : ξ_i is the state of the abstract object O_i. It is an interger in the range 1....RN, where, if $(k-1)N + 1 \leq \xi_i \leq kN$, then object O_i is assigned to α_k.

```
Method :  Initialize arbitrarily $\xi_i$ for $1 \le i \le W$, each class having W/R objects.
          Repeat
             For a sequence of T queries do
                ReadQuery ($A_i$, $A_j$)
                If (($\xi_i$ div N) = ($\xi_j$ div N)) Then (*The partitioning is rewarded*)
                   If (($\xi_i$ mod N) $\neq$ 1) Then   (*Move $O_i$ towards the internal
                      $\xi_i = \xi_i - 1$                      state*)
                   EndIf
                   If (($\xi_j$ mod N) $\neq$ 1) Then   (*Move $O_j$ towards the internal
                      $\xi_j = \xi_j - 1$                      state*)
                   EndIf
                Else      (* The partitioning is penalized *)
                   If ((($\xi_i$ mod N) $\neq$ 0) and (($\xi_j$ mod N) $\neq$ 0)) Then
                     (* Both are in internal states *)
                      $\xi_i = \xi_i + 1$
                      $\xi_j = \xi_j + 1$
                   Else If ($\xi_i$ mod N $\neq$ 0) Then   (* $O_i$ is in an internal state *)
                            $\xi_i = \xi_i + 1$
                   Else If ($\xi_j$ mod N $\neq$ 0) Then   (* $O_j$ is in an internal state *)
                            $\xi_j = \xi_j + 1$
                   Else      (* Both are in boundary states *)
                      temp = $\xi_i$        (* Store the state of $O_i$ *)
                      $\xi_i = \xi_j$            (* Move $O_i$ to same group as $O_j$ *)
                      t =  index of an unaccessed object in group of $O_j$
                             where $O_t$ is closest to $\xi_j$
                      $\xi_t$ = temp        (* Move $O_t$ to the old state of $O_i$ *)
                   EndIf
                EndIf
             EndFor
             Print out partitions based on the states {$\xi_i$}
          ForEver
End Algorithm OMA
```

IV.1 Analytical Results Concerning the OMA

A rigorous analysis of the OMA in terms of its convergence accuracies is by no means trivial. In its truest sense, the OMA is a strategy involving an interaction of the query system **and** the various objects, and thus in this perspective, it must be viewed as a co-operative automaton game. We are currently investigating the analytic properties of the probability of converging to the optimal partitioning. From our experience with learning automata and the experimental results we have obtained, we conjecture that the scheme is ε-optimal.

In the special case when $W = 4$ and $R = 2$, we shall now show that the OMA is expedient, even when N is as small as **unity.**

Theorem IV

When $W = 4$ and $R = 2$, the Object Migrating Automaton with $N = 1$ is expedient.

Proof :

Let the objects be A_1, A_2, A_3, A_4 , and let the probability of a query (A_i, A_j) be given as $Pr[(A_i, A_j)]$. We define the following quantities :

$$C_{12} = Pr\,[(A_1,A_2)] + Pr\,[(A_3,A_4)]$$
$$C_{13} = Pr\,[(A_1,A_3)] + Pr\,[(A_2,A_4)]$$
$$C_{14} = Pr\,[(A_1,A_4)] + Pr\,[(A_2,A_3)]$$

Note that $\Sigma_{2 \le i \le 4}\ C_{1i} = 1$. For the problem to be meaningful, there must be a partitioning superior to the others. Hence, one of the C_{1i}'s must be strictly greater than 1/3, and the sum of the other two strictly less than 2/3.

The markov chain that describes the OMA is thus given by three states $\{\omega_{12}, \omega_{13}, \omega_{14}\}$ where ω_{1i} represents the condition that A_1 should be grouped together with A_i. Notice that if i, j and k are distinct indices in the set {2, 3, 4}, then both the queries (A_1, A_i) and (A_j, A_k) support the partitioning represented by ω_{1i}. Thus the transition matrix of the markov chain of the OMA is given by the matrix F^*, where,

$$F^* = \begin{bmatrix} 1-C_{13}-C_{14} & C_{13} & C_{14} \\ C_{12} & 1-C_{12}-C_{14} & C_{14} \\ C_{12} & C_{13} & 1-C_{12}-C_{13} \end{bmatrix}$$

The chain is ergodic, and hence, using the theory of markov chains, the limiting state occupation probabilities obey the equation,

$$\underline{\Pi} = (F^*)^T \underline{\Pi} \qquad (8)$$

where $\underline{\Pi} = [\,\Pi_2, \Pi_3, \Pi_4\,]^T$, and $\Pi_i = Pr\,[\text{ state as } t \to \infty \text{ is } \omega_{1i}\,]$

The system of equations represented by (8) are :

$$\begin{array}{llllllll}
(-C_{13}-C_{14})\Pi_2 & + & C_{12}\Pi_3 & + & C_{12}\Pi_4 & = 0 \\
C_{13}\Pi_2 & + & (-C_{12}-C_{14})\Pi_3 & + & C_{13}\Pi_4 & = 0 \\
C_{14}\Pi_2 & + & C_{14}\Pi_3 & + & (-C_{12}-C_{13})\Pi_4 & = 0
\end{array}$$

Solving the above sets of equations and simplifying yields :

$$\Pi_i = C_{1i} / (C_{12} + C_{13} + C_{14}) \tag{9}$$

Observe that the OMA is expedient if the limiting loss is strictly less than the initial loss. Using (9), we write the expression for the limiting loss, $M(\infty)$, as :

$$M(\infty) = \frac{(C_{13}+C_{14})C_{12} + (C_{12}+C_{14})C_{13} + (C_{12}+C_{13})C_{14}}{C_{12} + C_{13} + C_{14}} \tag{10}$$

Similarly, if each action is initially chosen with probability 1/3, we write,

$$M(0) = 1/3 \; [\, (C_{13} + C_{14}) + (C_{12} + C_{14}) + (C_{12} + C_{13}) \,] \tag{11}$$

Comparing (10) and (11), we aim to prove that

$$\frac{2\,(C_{12}C_{13} + C_{12}C_{14} + C_{12}C_{13})}{C_{12} + C_{13} + C_{14}} < \frac{2\,(C_{12} + C_{13} + C_{14})}{3}$$

or in other words, that

$$3(C_{12}C_{13} + C_{12}C_{14} + C_{12}C_{13}) < (C_{12} + C_{13} + C_{14})^2 \tag{12}$$

But $\quad C_{12} + C_{13} + C_{14} = 1$. Thus, the OMA is expedient if

$$C_{12}C_{13} + C_{13}C_{14} + C_{12}C_{14} < 1/3. \tag{13}$$

But, by Lemma I and the fact that all the C_{1i}'s cannot be identically equal to 1/3, (13) is true. Hence the result. • • •

Lemma I

Let p, q, $r \geq 0$, such that $p+q+r = 1$. Then, $pq + qr + rp$ has a maximum value of 1/3, this value being achieved when $p = q = r = 1/3$.

Proof :

Let $f(p,q) = pq + qr + rp$

$$= pq + (p + q)\,(1 - (p+q))$$

We maximize $f(\cdot)$ by taking partial derivatives with respect to p and q. Thus,

$$\partial f(\cdot) / \partial p = 0 \Rightarrow 2p + q = 1 \tag{14}$$

Similarly, $\quad \partial f(\cdot) / \partial q = 0 \Rightarrow 2q + p = 1 \qquad (15)$

Solving (14) and (15) yields the result that $f(\cdot)$ is maximized when $p = q = r = 1/3$. Further the maximum value of $f(\cdot)$ is exactly 1/3. Hence the lemma. ● ● ●

From the above, we observe that for larger values of W and R, the proof concerning the ε-optimality (or the sub-optimality) of the scheme will definitely be not-trivial.

IV.2 Experimental Results

The BAM and the OMA were compared experimentally for various values of W and R. In the case of the BAM, as explained in Section III.1, the X_i's were uniformly distributed in [0,100] and Δ_1 and Δ_2 were set to unity. For the OMA, the number of states per action was set to 10, and the notion of convergence used was exactly as dscribed earlier. It must be stated in all fairness, that by processing an initial set of queries, the X_i's, Δ_1 and Δ_2, defined for the BAM, can be appropriately chosen so as to catalyze the convergence. But, obviously this can be used to hasten the convergence of the automata solutions too. By processing an initial set of queries, the ξ_i's (the states occupied by the abstract objects) can be initially assigned so as to yield very fast convergence. To render the comparison fair, we have completely avioded the processing of any statistics **even** in the initialization stage.

A summary of the results obtained are found in Table II. From the table, we observe the superiority of the OMA to the BAM. For example, when W, the number of objects, is 18 and R = 2, on the average, the BAM reached the optimal solution after 3690 iterations, and it converged to this solution after 3890 iterations. The OMA recorded the corresponding figures as 60 and 200 respectively. Notice that this implies an improvement by a factor of almost 19.5. Further, the OMA provided correct grouping in all the experiments performed. But in case of the BAM, when W = 12 and R = 2, among the ten experiments performed, only two gave the expected grouping, and the other eight did not. In these failed experiments, one of the expected groups was actually partitioned into two sub-groups. Similar situation was also reported in [30]. A comparative graphical sketch of the average number of iterations required for convergence of the BAM and the OMA is drawn in Figure IV as a function of the number of objects. In this case, for each point on the graph the number of objects per group is three. The advantage of the OMA is obvious.

No. Objects	Objects/Class	No. Classes	# iterations BAM	# iterations OMA
4	2	2	(50, 80)	(10, 45)
6	2	3	(200, 240)	(8, 65)
	3	2	(710, 770)	(5, 55)
9	3	3	(540, 650)	(50, 140)
12	2	6	(1550, 1630)	(30, 140)
	3	4	(910, 1000)	(90, 180)
	4	3	(1010, 1100)	(120, 210)
	6	2	(1470, 1570)*	(140, 230)
			(6480, 6530)**	
15	3	5	(1240, 1320)	(130, 260)
	5	3	(1270, 1330)	(130, 220)
18	2	9	(3690, 3890)	(60, 200)
	3	6	(1850, 2060)	(140, 280)
	6	3	(2070, 2190)	(210, 390)
	9	2	(710, 810)	(250, 410)

Table II : Comparison of the BAM and the OMA solution of the OPP. In the latter case the depth of memory per action is 10. In all cases, p, the probability defined by (6) is 0.9.

Notation : Number of iterations is given as a pair (x,y), where x is the average number of iterations for the grouping to be obtained, and y, the average number of iterations for the algorithm to converge to the grouping.

Note :

* Based on the average of the eight experiments which did not give the correct grouping.

** Based on the average of the two experiments which gave the correct grouping.

From our experimental experience we make the following comparative remarks regarding the various solutions to the OPP :

(i) For small W, when p is close to 1 (say 0.8 or greater), the OMA is almost as fast as the Tsetlin automaton. But when p is small, the latter is much faster. The same comment is true for the Krinsky automaton. Further the Krinsky automaton tends to be more stable than the Tsetlin automaton.

(ii) When W is large, the Tsetlin and Krinsky automata are not too feasible, but are still much faster than the OMA. The latter is easy to program than the former two, and it yet converges an order of magnitude faster than the BAM.

(iii) Although the BAM is slower than the techniques we presented, it is still the best known technique for the case when the set $\mathcal{A}$ need not be equi-partitioned. As it stands now, the OMA cannot handle the case when all the classes have a (possibly) different number of objects. We are currently investigating generalizations of our scheme to handle this scenario.

To conclude this section we present some experimental evidence to demonstrate the power of the OMA when the queries may involve more than two objects. Our strategy to handle a J-ary query is to consider it as ${}_JC_2$ "sub-query" inputs to the OMA which manipulates the abstract objects. In this case, the "sub-queries" access exactly two distinct abstract objects. Table III displays the experimental results we have obtained.

V. CONCLUSION

In this paper we have considered the problem of partitioning a set of objects $\mathcal{A} = \{A_1, \ldots, A_W\}$ into R classes of equal size. The aim is to partition $\mathcal{A}$ in such a way that the objects accessed together are found in the same class. We present three deterministic automata solutions to the problem. The first two use the well known Tsetlin and Krinsky automata, and thus have ε-optimal properties. The last one, uses a new scheme which we have called the Object Migrating Automaton (OMA). In a fairly simple example this automaton is shown to be expedient. However, simulation results seem to indicate its ε-optimality. Further, these results prove that it converges an order of magnitude faster than the best known algorithm in the literature.

No. Objects	Objects/Class	No. Classes	# iterations OMA
9	3	3	(20, 55)
12	3	4	(50, 60)
	4	3	(45, 85)
15	3	5	(60, 100)
	5	3	(55, 125)
18	3	6	(70, 150)
	6	3	(60, 130)

Table III : Results demonstrating the power of the OMA to solve the Object Partitioning Problem when the number of objects accessed per query is 3. In this case p, defined by (6) is 0.9.

Notation : Same as Table I and II.

REFERENCES

1. Arnow, D.M., and Tenenbaum, A. M., "An Investigation of the Move-Ahead-k-Rules", Congressus Numerantium, Proc. of the Thirteenth Southeastern Conference on Combinatorics, Graph Theory and Computing, Florida, Feb. 1982, pp.47-65.

2. Bitner, J.R.,"Heuristics that Dynamically Organize Data Structures", SIAM J. Computing, Vol. 8, 1979, pp. 82-110.

3. Gonnet, G.H., Munro, J.I., and Suwanda, H., "Exegesis of Self Organizing Linear Search", SIAM J. Computing, Vol. 10, 1981, pp. 613-637.

4. Hammer, M., and Niamir, B., "A Heuristic Approach to Attribute Partitioning", Proc. of the ACM SIGMOD Conference, 1979, pp.93-101.

5. Hammar, M., and Chan, A., "Index Selection in a Self-adaptive Database Management System", ACM SIGMOD Conference, 1976, pp.1-8.

6. Hendricks, W. J.,"An Account of Self-Organizing Systems", SIAM J. Computing, Vol. 5, 1976, pp. 715-723.

7. Kan, Y. C., and Ross, S.M., "Optimal List Order Under Partial Memory Constraints", J. App. Probability, Vol. 17, 1980, pp. 1004-1015.

8. Knuth, D.E., "The Art of Computer Programming, Vol. 3, Sorting and Searching", Addison-Wesley, Reading, MA, 1973.

9. Lakshmivarahan, S. and Thathachar, M.A.L., "Absolutely Expedient Algorithms for Stochastic Automata", IEEE Trans. on Syst, Man and Cyber., Vol. SMC-3, 1973, pp.281-286.

10. Lakshmivarahan, S., "Learning Algorithms Theory and Applications", Springer-Verlag, New York, 1981.

11. Lam, K. and Yu, C.T., "A Clustered Search Algorithm Incorporating Arbitrary Term Dependencies", Dept. of Information Eng., Univ. of Illinois at Chicago Circle, 1978.

12. Lam, K. and Yu, C.T., "An Approximation Algorithm for a File Allocation Problem in a Hierarchical Distributed System", ACM SIGMOD Conference, 1980, pp.125-132.

13. McCabe, J., "On Serial Files with Relocatable Records", Operations

Research, Vol. 12, 1965, pp.609-618.

14. Narendra, K.S., and Thathachar, M.A.L., "Learning Automata -- A Survey",IEEE Trans. Syst. Man and Cybern., Vol.SMC-4, 1974, pp. 323-334.

15. Narendra, K.S., and Thathachar, M.A.L., "On the Behaviour of a Learning Automaton in a Changing Environment With Routing Applications", IEEE Trans. on Syst. Man and Cybern., Vol.SMC-10, 1980, pp. 262-269.

16. Narendra, K.S., Wright, E. and Mason, L.G., "Application of Learning Automata to Telephone Traffic Routing", IEEE Trans. on Syst. Man and Cybern., Vol.SMC-7, 1977, pp.785-792.

17. Oommen, B.J., and Hansen, E.R., "The Asymptotic Optimality of Two Action Discretized Linear Reward-Inaction Learning Automata", IEEE Trans. on Systems, Man and Cybernetics, May/June 1984, pp. 542-545.

18. Oommen, B.J., "Absorbing and Ergodic Discretized Two Action Learning Automata", To appear in IEEE Trans. on Systems, Man and Cybernetics.

19. Oommen, B.J.,"On the Use of Smoothsort and Stochastic Move-to-Front Operations for Optimal List Organization", Proc. of the Twenty-Second Allerton Conference, Oct. 1984, Urbana, Illinois, pp.243-252.

20. Oommen, B.J., and Hansen, E.R.,"List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations", Submitted for Publication. Also available as SCS-TR-76, from Carleton University, Ottawa.

21. Rivest, R.L.,"On Self Organizing Sequential Search Heuristics", Comm. ACM, Vol. 19, 1976, pp. 63-67.

22. Sleator, D. and Tarjan, R., "Amortized Efficiency of List Update Rules", Proc. of the Sixteenth Annual ACM Symposium on Theory of Computing, April 1984, pp.488-492.

23. Salton, G., "Dynamic Information and Library Processing", Prentice Hall, Englewood Cliffs, New Jersey, 1975.

24. Schkolnick, M., " A Clustering Algorithm for Hierarchical Structures", ACM Trans. on Database Systems, 1977, pp.27-44.

25. Tsetlin, M.L., "On the Behaviour of Finite Automata in Random Media", Automat. Telemek. , Vol. 22, 1961, pp.1345-1354.

26. Tsetlin, M.L.,"Automaton Theory and Modelling of Biological Systems", Academic Press, New York and London, 1973.

27. Van Rijsbergen, C.J., "A Theoretical Basis for the Use of Co-occurrence Data in Information Retrieval", J. Documentation, 1977, pp.106-119.

28. Yu, C.T., and Salton, G., "Precision Weighting - An Effective Automatic Indexing Method", J. ACM, 1976, pp.76-78.

29. Yu, C.T., Siu, M.K., Lam, K., and Ozsoyoglu, M., "Performance Analysis of Three Related Assignment Problems", Proc. of the ACM SIGMOD Conference, May 1979.

30. Yu, C.T., Siu, M.K., Lam, K., and Tai, F., "Adaptive Clustering Schemes : General Framework", Proc. of the IEEE COMPSAC Conference, 1981, pp.81-89.

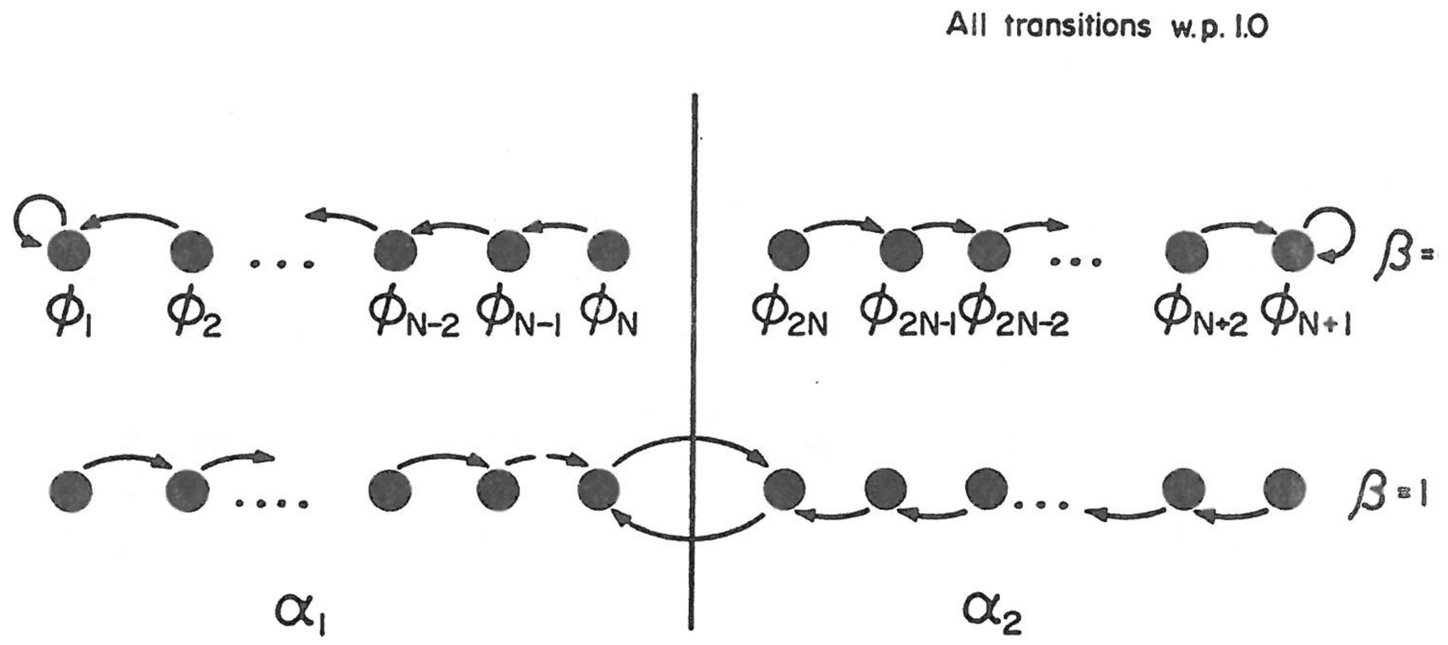


FIGURE I: $L_{2N,2}$, THE TWO-ACTION 2N-STATE TSETLIN AUTOMATON

All transitions w.p. 1.0

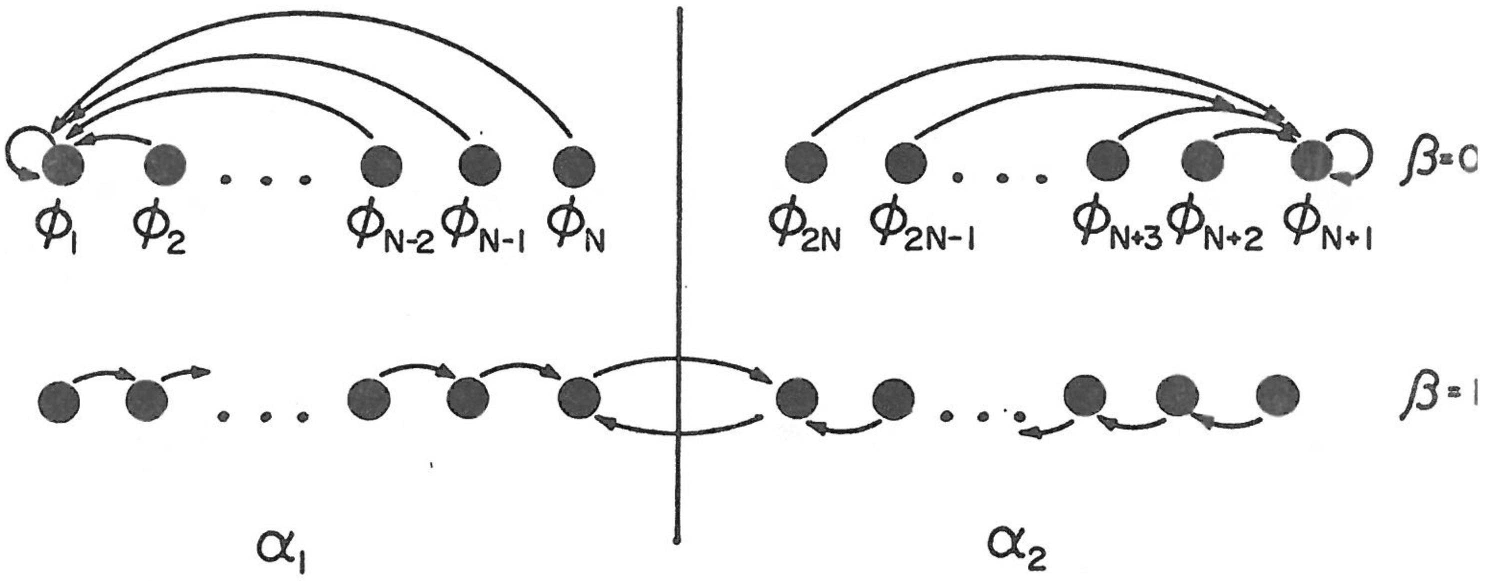
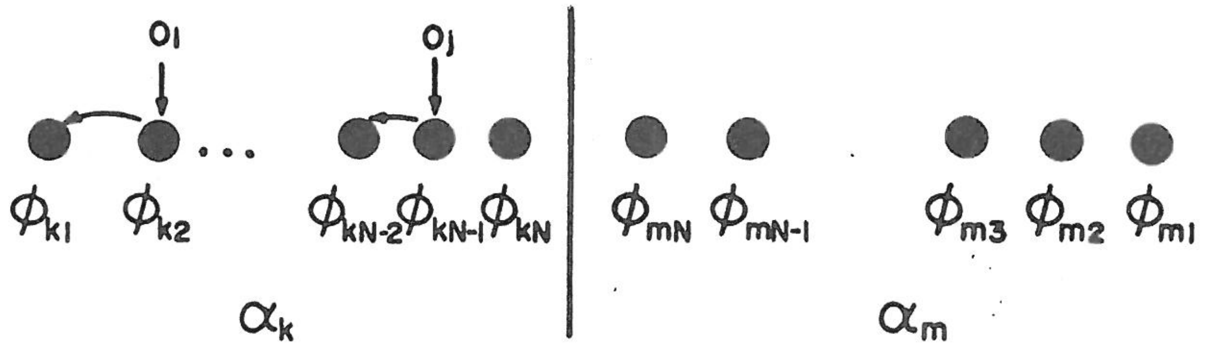
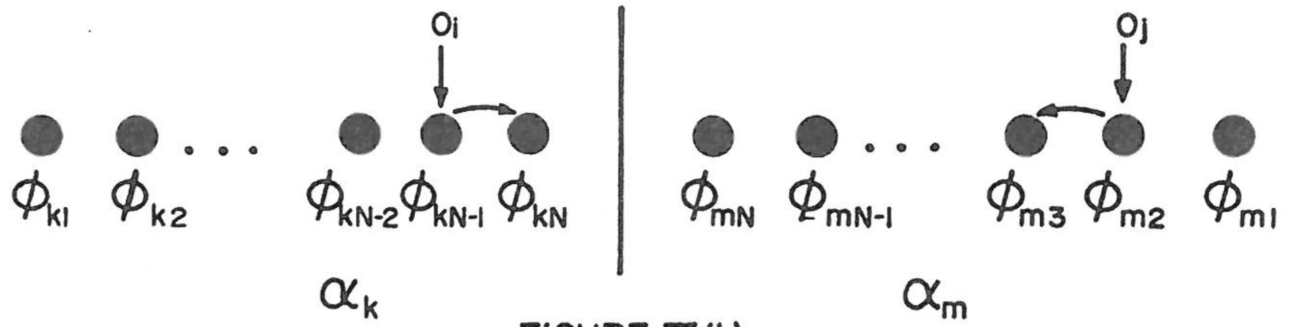


FIGURE II: $Z_{2N,2}$, THE TWO-ACTION 2N-STATE KRINSKY AUTOMATON



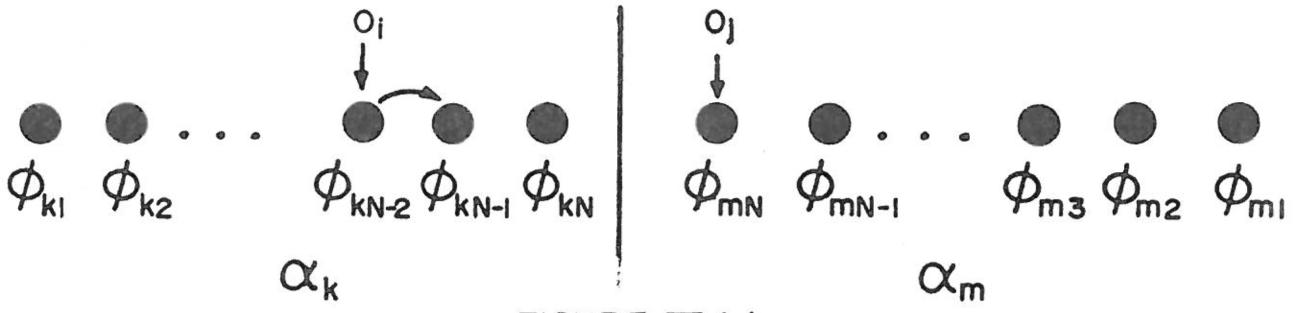
O_i
O_j
ϕ_{k1} ϕ_{k2} ϕ_{kN-2} ϕ_{kN-1} ϕ_{kN}
ϕ_{mN} ϕ_{mN-1} ϕ_{m3} ϕ_{m2} ϕ_{m1}
α_k
α_m

FIGURE III (a)



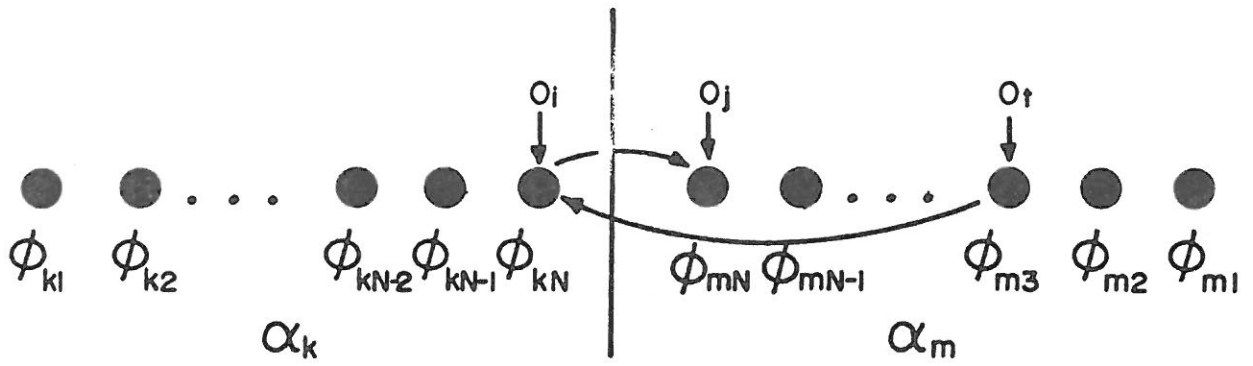
O_i
O_j
ϕ_{k1} ϕ_{k2} ϕ_{kN-2} ϕ_{kN-1} ϕ_{kN}
ϕ_{mN} ϕ_{mN-1} ϕ_{m3} ϕ_{m2} ϕ_{m1}
α_k
α_m

FIGURE III (b)



O_i
O_j
ϕ_{k1} ϕ_{k2} ϕ_{kN-2} ϕ_{kN-1} ϕ_{kN}
ϕ_{mN} ϕ_{mN-1} ϕ_{m3} ϕ_{m2} ϕ_{m1}
α_k
α_m

FIGURE III (c)



O_i
O_j
O_t
ϕ_{k1} ϕ_{k2} ϕ_{kN-2} ϕ_{kN-1} ϕ_{kN}
ϕ_{mN} ϕ_{mN-1} ϕ_{m3} ϕ_{m2} ϕ_{m1}
α_k
α_m

FIGURE III (d)

FIGURE III (a)(b)(c)(d) The object moving Automaton: On reward move the abstract objects to the most internal state for that action as in (a). On penalty move the abstract objects either toward the boundary state as in (b) or (c), or toward the boundary state of another action (d).

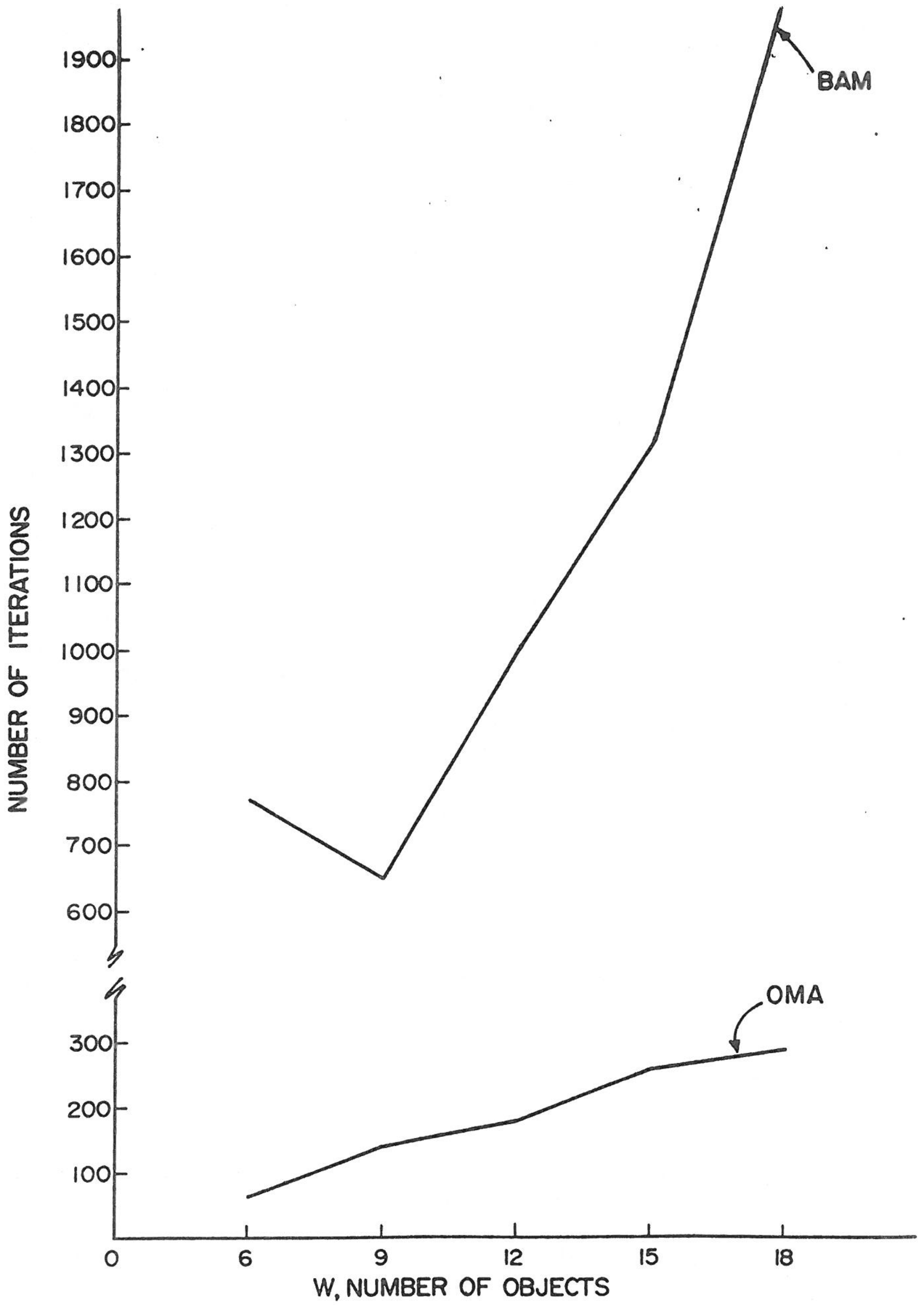


FIGURE IV: Plot of average number of iterations required for convergence for the BAM and the OMA as the number W increases. In each case the number of objects per group is 3.