# STOCHASTIC AUTOMATA SOLUTIONS TO THE OBJECT PARTITIONING PROBLEM

B.J. Oommen
D.C.Y. Ma

SCS-TR-103
November 1986

School of Computer Science
Carleton University
Ottawa, Ontario
CANADA K1S 5B6

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada.

# Stochastic 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 $\Omega = \{A_1, \ldots, A_W\}$ be a set of W objects to be partitioned into R classes $\{P_1,\ldots,P_R\}$. The objects are accessed in groups of unknown size and the size of these groups need not be equal. Additionally, the joint access probabilities of the objects are unknown. The intention is that the objects accessed more frequently together are located in the same class. This problem has been shown to be NP-hard [30,31]. In this paper, we propose two stochastic learning automata solutions to the problem. Although the first one is relatively fast, its accuracy is not so remarkable in some environments. The second solution, which uses a new variable structure stochastic automaton, demonstrates an excellent partitioning capability. Experimentally, this solution converges an order of magnitude faster than the best known algorithm in the literature [30,31].

**Keywords :** Learning Systems, Applications of learning, Adaptive Learning, Object Partitioning, Distributed File Allocation, Intelligent Partitioning of Attributes and Records.

---

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

## I. INTRODUCTION

Let $\Omega = \{A_1, ..., A_W\}$ be a set of W objects. We intend to partition $\Omega$ into R classes $\{P_1, ..., P_R\}$. 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). To render the problem non-trivial, we assume that their joint access probabilities are unknown.

At the very outset, we note that the basic assumption for this study is that there is an unknown underlying partition of the objects. This, in a sense, is determined by the user's query (or access) patterns. We shall refer to this partitioning of the objects as the **grouping** imposed on them. Observe that although this grouping is unknown, our aim is to partition the objects into classes that **mimic** the underlying **grouping**. For the rest of the paper, we shall use the words class and partition interchangeably. We state in passing that the Object Partitioning Problem (OPP) has been shown to be NP-hard [30,31] due to the exponential growth in the number of partitions of the objects.

There are various applications for the OPP. In a library environment [23], the set $\Omega$ 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 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). Some other computer science applications of the OPP are :

a) Let $\Omega$ be a set of attributes belonging to a normalized relation. Because of the size of $\Omega$, 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 $\Omega$ be a set of records stored on a disk. We wish 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, can segment clustering in IMS trees [30,31] be considered as a record clustering problem [24].

c) In a distributed data base, the files that are distributed can be located at

various sites. In this case, it is desirable that files which are jointly accessed are physically located in the same geographical site. Obviously this would minimize communication costs [12,29].

Other applications of the OPP such as the selection of dependent terms and the automatic indexing problem are also 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 of solving 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, the use of such statistics is impractical due to extensive 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 descending order of their access probabilities [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 actually keeping a count on the number of times they are accessed (i.e. estimating their access probabilities). The excessive memory required with this approach, however, motivated the search for self-organizing algorithms such as the Move-to-Front, the Transposition, the Move-k-Ahead Rule etc. [3]. Unfortunately, the solutions for the list organizing problem are not directly usable to solve the OPP 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* [30,31]. 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. In its time it probably was the best algorithm to solve the attribute partitioning problem. In [30,31], Yu *et al* proposed an adaptive scheme which was truly ingenious. This technique is called the Basic Adaptive Method (BAM). We shall describe this scheme in a later section, but we comment in passing that the authors of [30,31] demonstrate the superiority of their algorithm to the one in [4]. We refer the reader to [30,31] for a superb review and comparison of the various algorithms.

In this paper we shall present some learning automata solutions to the OPP. Learning automata have been used in the literature to model biological learning systems and also to determine the optimal action which an environment offers. The learning is achieved by the automaton 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 automata of the latter type 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].

For the rest of this section we briefly discuss the fundamentals of learning automata. In the next section we present the algorithm due to Yu *et al* [30,31] 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 of Learning Automata

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

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

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

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

(iv) F is a map from $\Phi \times \beta$ to $\Phi$. It defines the state transition of the automaton on

receiving an input. F may be stochastic.

(v) G is a map from $\Phi$ to $\alpha$, 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 response of the environment which is fed back to the automaton. The environment penalizes (i.e., $\beta(n) = 1$) the automaton with the penalty $c_i$, where,

$$c_i = \Pr[\beta(n) = 1 \mid \alpha(n) = \alpha_i] \qquad (i = 1,..,R).$$

Thus the environment characteristics are specified by the set of penalty probabilities $\{c_i\}$ $(i = 1,..,R)$. 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.

A **Variable Structure Stochastic Automaton** (VSSA) is essentially [14] an updating rule by which the action probability vector at the $(n+1)^{st}$ time instant is computed using the probability vector at the $n^{th}$ time instant and the automaton-environment interaction at the $n^{th}$ time instant. Thus, if $\mathbf{P}(n)$ is the action probability vector at time 'n', the VSSA is fully defined by specifying a function H, where, $\mathbf{P}(n+1) = H(\mathbf{P}(n), \alpha(n), \beta(n))$.

## I.2 Learning Criteria

With no *a priori* 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 **$\varepsilon$-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.

### 1.3 Summary of results obtained

The first attempt to solve the OPP using stochastic automata was presented by us [19] for a simplified version of the OPP in which the underlying groups of the objects were of the same size. This problem has been called the Equal Partitioning Problem (EPP). Three deterministic automata solutions to the EPP were presented in [19]. The first two solutions which involved the Tsetlin and Krinsky automata are both fast and accurate but are feasible only when the number of objects to be partitioned is small. The last automaton described in [19] was the Object Migrating Automaton (OMA) [19]. For a special set of objects, it was shown to be expedient in all random environments. Experimental results involving the OMA demonstrated that it is as accurate as the technique due to Yu *et. al.* [30,31] and is yet an order of magnitude faster than the latter for the EPP.

In this paper we shall consider the OPP in greater generality than in [19]. In other words, we shall not assume that the underlying groups are of equal size. We shall first study the use of the Object Migrating Automaton (OMA) to solve the OPP. We shall then present another fixed structure automaton, the Stochastic Move Automaton (SMA), and demonstrate its capability in solving the OPP. The latter automaton converges much faster than the method due to Yu et al [30,31] although in terms of accuracy it is inferior in some environments. Finally, we shall present a new **variable** structure stochastic automaton that is quite similar to the traditional Linear Reward Penalty Scheme. This automaton, termed as the Modified Linear Reward Penalty ($ML_{RP}$) scheme, is shown to be expedient in all random environments. Experimentally, it has a remarkable accuracy in that it converges to the right partitioning almost **all** the time. Yet it is, on the average, about ten times faster (and sometimes even about twenty times faster) than the BAM method due to Yu *et. al.*

In the next section, we present the BAM due to Yu *et. al.* , which, in our opinion, is the best known technique to solve the OPP. We shall then present our automata solutions to the problem.

## II. THE BASIC ADAPTIVE METHOD

Yu *et al* [30,31] proposed an algorithm to solve the OPP, which they have called the Basic Adaptive Method (BAM). In our reckoning, this is the best known algorithm for the OPP, and its advantages over the technique due to Hammer *et . al.* [14] are well illustrated in [30,31]. We describe below the BAM in the situation when exactly two objects are accessed per query. (The case when more than two objects are accessed has also been handled in [30,31]).

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 $\Omega = \{A_1, A_2, ..., A_W\}$ 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 $\Omega$ are re-partitioned so as to conform to the partitioning dictated by the BAM. This periodic repartitioning of $\Omega$ 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 $\Delta_1$. This tends to group the points into clusters. 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 $\Delta_2$. The techniques of "determining" $\Delta_1$, $\Delta_2$ and the initial $X_i$'s are informally discussed in [30,31]. 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 [30,31]. 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 $\Delta_1$ and $\Delta_2$, 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 $1 \leq i \leq W$, 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 $1 \leq p, q \leq W$

**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 a FSSA solution to the OPP. However, before we study the Stochastic Move Automaton (SMA), to add to the paper's continuity, we shall **briefly** discuss the Object Migrating Automaton introduced in [19]. For the sake of brevity, the results pertaining to the OMA will merely be alluded to whenever these results are available in the literature.

## III. 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 Tsetlin and Krinsky automata, [25] 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 $\alpha_k$, this signifies 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 $\alpha_k$, $1 \leq k \leq R$, there is a set of states $\{\phi_{k1}, ..., \phi_{kN}\}$, where N is the depth of memory and R is the number of classes (partitions) desired. With no loss of generality, we assume $\phi_{k1}$ to be the most internal state of that action and $\phi_{kN}$ 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 $\alpha_k$, both of them are rewarded and are moved toward the most internal state, $\phi_{k1}$, of that action, one step at a time. See Figure I(a). If, however, the abstract objects lie in different classes, say $\alpha_k$ and $\alpha_m$ respectively, (i.e. $O_i$ is in state $\xi_i$, where $\xi_i \in \{\phi_{k1}, ..., \phi_{kN}\}$, and $O_j$ is in state $\xi_j$, where $\xi_j \in \{\phi_{m1}, ..., \phi_{mN}\}$), they are moved **away** from $\phi_{k1}$ and $\phi_{m1}$ 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 $\phi_{kN}$ and $\phi_{mN}$, respectively. See Figure I (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 I (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 $\alpha_m$. In this case, since this will result in an excess of objects in $\alpha_m$, one of the objects in $\alpha_m$ which is **not** accessed, is moved to the boundary state of $\alpha_k$. We choose to move the object closest to $\xi_j$. See Figure I(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. For the sake of completeness, we now algorithmically give the above described technique. For simplicity we merely update the indices of the states in which the objects are in.

**Algorithm OMA**

**Input** : The abstract objects $\{O_1, ..., O_W\}$ and N, the number of states 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** : $\xi_i$ is the state of the abstract object $O_i$. It is an integer in the range

1....RN, where, if $(k-1)N+1 \leq \xi_i \leq kN$, then object $O_i$ is assigned to $\alpha_k$.

**Method :** Initialize arbitrarily $\xi_i$ for $1 \leq i \leq W$, each class having W/R objects.

```
Repeat
    For a sequence of T queries do
        ReadQuery (Ai, Aj)
        If ((ξi div N) = (ξj div N)) Then (*The partitioning is rewarded*)
            If ((ξi mod N) ≠ 1) Then    (*Move Oi towards the internal
                ξi = ξi - 1                      state*)
            EndIf
            If ((ξj mod N) ≠ 1) Then    (*Move Oj towards the internal
                ξj = ξj - 1                      state*)
            EndIf
        Else     (* The partitioning is penalized *)
            If (((ξi mod N) ≠ 0) and ((ξj mod N) ≠ 0)) Then
              (* Both are in internal states *)
                ξi = ξi + 1
                ξj = ξj + 1
            Else If (ξi mod N ≠ 0) Then   (* Oi is in an internal state *)
                        ξi = ξi + 1
            Else If (ξj mod N ≠ 0) Then   (* Oj is in an internal state *)
                        ξj = ξj + 1
            Else     (* Both are in boundary states *)
                temp = ξi       (* Store the state of Oi *)
                ξi = ξj         (* Move Oi to same group as Oj *)
                t =  index of an unaccessed object in group of Oj
                        where Ot is closest to ξj
                ξt = temp       (* Move Ot to the old state of Oi *)
            EndIf
        EndIf
    EndFor
    Print out partitions based on the states {ξi}
ForEver
End Algorithm OMA
```

### III.1 Analytical Results Concerning the OMA

A rigorous analysis of the OMA in terms of its accuracy of convergence 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 is indeed a co-operative automaton game. From our experience with learning automata and the experimental results for the EPP [19], we conjecture that the scheme is $\varepsilon$-optimal.

In the special case when W = 4 and R = 2, it has been shown that the OMA is expedient for the EPP, even when N is as small as **unity**.

**Theorem I**

When W = 4 and R = 2, the Object Migrating Automaton with N = 1 is expedient. The theorem is proved elsewhere [19]. ♦ ♦ ♦

We refer the reader to [19] for a more detailed description of the OMA and for examples clarifying its operation. Also found elsewhere [19, 32] are experimental results demonstrating its superiority to the BAM for the equi-partitioning problem. We shall now proceed to study the OPP.

### III.2 Generalization of the OMA Approach for the OPP

Compared to the Tsetlin and Krinsky automata, generalizing the OMA to handle the OPP is certainly non-trivial because the main concept which justifies the operation of the OMA is the limit imposed on the number of objects in each class. This limit is used to prevent two or more groups of objects from coalescing. In the case of the EPP, this limit is obviously W/R, when W objects are to be partitioned into R classes. In the case of the OPP, however, the value W/R can no longer be used since each class may or may not have exactly W/R objects in it.

Due to the fact that each class may have a different number of objects, one possible solution is that of setting an **upper bound** $\Sigma$ on the class size instead of a fixed limit on it (i.e., at each time instant, each class can have **at most** $\Sigma$ objects). Just like the OMA discussed previously, if an object $O_i$ leaves action $\alpha_i$ and enters action $\alpha_j$ which already has $\Sigma$ objects, then one of the objects in $\alpha_j$ not being accessed will be moved to $\alpha_i$.

Consider the following elementary question: Can the use of an upper bound on the class size prevent different groups of objects from entering the same class? If the answer to the question were in the affirmative, this would lead to a possible solution of the OPP. However, it can be shown that this is not the case.

**Theorem 2**

In the OMA, setting an upper bound on the class size does not **always** prevent more than one group of objects from entering the same class.

**Proof :**

Suppose W objects are given and these objects are partitioned into R classes, not necessarily of equal size, based on the queries against them. The query statistics are, of course, unknown to the OMA. Consider the query system in which the queries support the partitioning where the group $\{O_1, ..., O_m\}$ is in one class, and $\{O_{m+1}, ..., O_{m+n}\}$ is in another class. We denote the former group as $G_1$, of size m (i.e., $|G_1| = m$), and the latter group as $G_2$, of size n (i.e., $|G_2| = n$), where m, n > 0. With no loss of generality, m,n ≥ 2. Further, suppose the queries also support the group $G_3 = \{O_{m+n+1}, ..., O_{2m+2n}\}$, of size m+n. Since no *a priori* information is given, we assume that every class has the same upper bound Σ, where Σ ≥ m+n.

Suppose at the $k^{th}$ time instant $O_i$, $O_j \in G_1$ are in classes (represented by actions) $\alpha_p$ and $\alpha_q$, respectively. In addition, all the n objects of $G_2$ are also in $\alpha_q$ (i.e., $\alpha_q$ has n+1 objects). Consider the situation when $O_i$ and $O_j$ are in the boundary states of $\alpha_p$ and $\alpha_q$, respectively. If the next query requests the pair ($A_i$, $A_j$), then according to the transition map of the OMA, either $O_i$ or $O_j$ will move to the action which contains the other object. With no loss of generality, $O_i$ is the one to leave $\alpha_p$ and enter $\alpha_q$. Adding $O_i$ to $\alpha_q$ makes $\alpha_q$ contain n+2 objects. Since n+2<Σ (because 2 ≤ m and m+n < Σ) and $\alpha_q$ can have at most Σ objects, no object in $\alpha_q$ is required to move to $\alpha_p$.

In this case, if m = 2, then we have both $G_1$ and $G_2$ found in the same action, namely $\alpha_q$. Since this implies that the elements of $G_1$ and $G_2$ are in the same class, they will be rewarded quite frequently and will eventually reach the most internal state of $\alpha_q$ leading to an erroneous partition. Hence the theorem is proved.

♦ ♦ ♦

Thus, setting an upper bound on the class size in the OMA does not seem helpful in solving the OPP.

In the following sections, we shall propose another FSSA which we have called the Stochastic Move Automaton (SMA). This automaton converges very quickly. However, it turns out that this automaton cannot handle the OPP accurately in all scenarios. Apart from its value as a solution to the OPP, it definitely is a stepping stone to the development of the variable structure stochastic automaton which we shall discuss subsequently.

### III.3 Two Important Properties

Before proceeding to the discussion of the SMA, we would like to discuss two important properties which, in our opinion, are essential for finding automata solutions to the OPP.

**Theorem III**

Initially assigning more than one object to a class **could** result in having more than one group of objects in the same class.

**Proof :**

Suppose a set of W abstract objects $\{O_1, ..., O_W\}$ is given and R classes (represented by actions) are available for them with $R < W$. Since $R < W$ and each object has to be assigned to one class, obviously, there will be at least one class with more than one object. Consider the case when $O_i$ and $O_j$ are in action $\alpha_k$ and the queries access the pairs $(A_i, A_m)$ and $(A_j, A_n)$ more frequently than the pairs $(A_i, A_j)$, $(A_i, A_n)$ and $(A_j, A_m)$. In this case, if the abstract objects $O_m$ and $O_n$ eventually entered action $\alpha_k$, they could stay with $O_i$ and $O_j$. This implies that $O_i$, $O_j$, $O_m$ and $O_n$ are in the same class, namely $\alpha_k$. However, the fact that the pairs $(A_i, A_j)$, $(A_i, A_n)$ and $(A_j, A_m)$ are seldom accessed and the pairs $(A_i, A_m)$ and $(A_j, A_n)$ are frequently accessed indicates that $O_i$ and $O_m$ should be in one class, say $\alpha_p$, and $O_j$ and $O_n$ should be in another class $\alpha_q$, where $\alpha_p \neq \alpha_q$. However, by the merging of the abstract objects $O_i$, $O_j$, $O_m$ and $O_n$ into the same class $\alpha_k$, such a subpartitioning is impossible and the result follows. ♦ ♦ ♦

**Theorem IV**

One **improper** movement of an object in an automaton **can** result in having more than one group of objects found in the same class.

**Proof :**

Consider the case when a set of abstract objects $\{O_1, ..., O_W\}$ is given and each object is initially assigned to choose a different class (or action). This implies that initially, each action has only one object assigned to it. Let objects $O_i$, $O_j$ and $O_k$ be in classes $\alpha_p$, $\alpha_q$, and $\alpha_r$, respectively, where $1 \leq p, q, r \leq W$ and $p \neq q \neq r$. Suppose the queries access the pair $(A_i, A_j)$ more frequently than the pairs $(A_i, A_k)$ and $(A_j, A_k)$. Then objects $O_i$ and $O_j$ should lie in the same class and $O_k$ should lie in another class. In order to achieve this goal, either $O_i$ or $O_j$ or both are required to change their classes, simply because they are in different classes. Without loss of generality, we can assume that $O_i$ is the object about to change its class (i.e., from $\alpha_p$ to $\alpha_q$). However, to render the problem meaningful we have assumed that with

a non-zero probability the queries may access objects that are in different groups. If the pair $(A_i, A_k)$ is accessed instead of the pair $(A_i, A_j)$, then $O_i$ which is about to leave from $\alpha_p$ will enter $\alpha_r$ instead of $\alpha_q$. This can also happen to $O_j$ and will result in having $O_i$ and $O_j$ in $\alpha_r$, which is the class for another group of objects. Hence the theorem. ♦ ♦ ♦

Although it appears as if these two criteria concern two different aspects of the problem, in actuality, they both play important roles in finding solutions to the OPP because one affects the performance of the other. We shall now proceed to the discussion of the SMA which uses these two properties in its structure.

## IV THE STOCHASTIC MOVE AUTOMATON

In order to avoid encountering the initialization problem discussed in Theorem II, the following method is proposed: Instead of assigning W objects to R classes, R < W, such that each class may have more than one object in it, the W objects are initially assigned to W **distinct** classes so that no two objects are found in the same class. Since all the objects are in different classes, if there is no "incorrect" movement during the process of partitioning objects, then obviously no two groups of objects will be found in the same class.

On the other hand, moving objects around in an automaton is not trivial due to the fact that the accesses are stochastic, i.e. that objects that actually should be in different groups may also be accessed together. Observe that such queries can be regarded as "incorrect" because they provide misleading information for objects that are moving around in the automaton. Although the existence of such queries is certain, it is **impossible** to detect them in advance, simply because the ideal grouping of the objects is unknown *a priori*. Hence, the best we can do is to hope to "live with them".

Suppose we come to a partitioning for which no object is required to change its action. Clearly, in this case, an "incorrect" query will not affect this partitioning. However, if we do not come to such a partitioning, according to Theorem IV, such a query may have adverse effects on the outcome, namely it can lead to an incorrect partitioning of the objects. To avoid this, one reasonable way is to try to ignore such a query. Alternatively, the abstract objects can be given the option of either following the information given by the query or ignoring the information. This is achieved as follows : An object $O_i$ , in the boundary state of an action may change

classes on receiving a penalty, from class $\alpha_p$ to class $\alpha_q$. The latter transition is performed with a probability of $\delta$. Additionally, it stays in $\alpha_p$ with probability $1 - \delta$. Specifically,

$$\Pr[\, O_i \text{ moves out from } \alpha_p \,] = \delta \tag{8}$$

$$\Pr[\, O_i \text{ stays in } \alpha_p \,] = 1 - \delta \tag{9}$$

The SMA employs these strategies to handle the OPP.

### IV.1 Structure of the SMA

Let $\Omega = \{A_1, ..., A_W\}$ be the set of W given objects. Just like the OMA, the set of abstract objects are moving around in the SMA. An SMA is defined as a quintuple $(\alpha, \Phi, \beta, F, G)$, where

(i) $\alpha = \{\alpha_1, ..., \alpha_W\}$ is the set of actions that the abstract objects must choose from. Each action $\alpha_i$, $1 \le i \le W$, represents a certain class.

(ii) $\Phi = \{\phi_1, ..., \phi_{WN}\}$ is the set of WN states. Each action $\alpha_p$ has N states associated with it. With no loss of generality, we can assume $\phi_{(p-1)N+1}$ is the most internal state and $\phi_{pN}$ is the boundary state of $\alpha_p$.

(iii) $\beta = \{0, 1\}$ is the set of inputs from the environment. The value '0' represents a reward and the value '1' represents a penalty.

(iv) Let $\xi_i$ and $\xi_j$ be the states of the abstract objects $O_i$ and $O_j$, respectively, and let the query be $(A_i, A_j)$. Then the transition matrix F is defined as follows:

$$\beta = 0 \quad \begin{cases} \xi_k = \xi_k - 1 & \text{if } (\xi_k \bmod N) \neq 1 \quad \text{for } k = i, j \quad (10) \\ \xi_k = \xi_k & \text{if } (\xi_k \bmod N) = 1 \quad \text{for } k = i, j \quad (11) \end{cases}$$

$$\beta = 1 \quad \begin{cases} \xi_k = \xi_k + 1 & \text{if } (\xi_k \bmod N) \neq 0 \quad \text{for } k = i, j \quad (12) \\ \xi_k = \lceil \xi_m / N \rceil * N & \text{if } (\xi_k \bmod N) = 0 \quad \text{for } k = i, j;\ m = i, j;\ k \neq m \quad (13) \end{cases}$$

The transitions (10), (11) and (12) are with probability 1.0, while the transition (13) is with probability $\delta$.

(v) If $O_i$ is in state $\xi_i$ and $\alpha_p$ is the action that $O_i$ is in, then G, the output matrix, is defined by the following equation:

$$p = \lceil \xi_i / N \rceil \qquad \text{for } 1 \le i \le W$$

Just like the OMA, the "query system" plays the role of the environment with which the SMA interacts. At each time instant, if $(A_i, A_j)$ is the pair of physical objects accessed and the corresponding abstract objects $O_i$ and $O_j$ are in action $\alpha_p$ and $\alpha_q$, respectively, then the environment will send out

(a) a reward (i.e., $\beta = 0$), if $\alpha_p = \alpha_q$

(b) a penalty (i.e., $\beta = 1$), if $\alpha_p \neq \alpha_q$

### IV.2 Algorithmic version of the SMA

Initially, each of the W abstract objects $O_i$ is assigned to an action $\alpha_k$, $1 \leq i, k \leq W$. Without loss of generality, we can assume $i = k$. At each time instant, a pair of physical objects $(A_i, A_j)$ is accessed. If the corresponding abstract objects $O_i$ and $O_j$ are in the same action, say $\alpha_k$, they will be rewarded and will move toward the most internal state of $\alpha_k$. On the other hand, if $O_i$ and $O_j$ are in different actions, say $\alpha_p$ and $\alpha_q$, then they will be penalized and will move toward the boundary states of $\alpha_p$ and $\alpha_q$, respectively. If either $O_i$ or $O_j$ or both need to change action, then the moving object(s) will move to another action with a probability $\delta$. Just like the OMA, if $O_i$ intends to leave $\alpha_p$ and enter $\alpha_q$, it can just move to $\alpha_q$ without going through the intermediate actions $\alpha_{(p+1) \bmod W} \cdot\cdot \alpha_{(q-1) \bmod W}$.

After T accesses, the physical objects are reorganized, if necessary, based on the actions chosen by the corresponding abstract objects. For the sake of completeness, we present below an algorithmic version of the SMA. As before, we only keep track of the states of the objects.

## Algorithm SMA

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

**Output** : A periodic partitioning of the objects.

**Notation** : $\xi_i$ is the state of the abstract object $O_i$. It is an integer in the range 1....WN, where, if $(k-1)N + 1 \leq \xi_i \leq kN$, then object $O_i$ is assigned to $\alpha_k$. The transition of $O_i$ and $O_j$ at the boundary states is determined by invoking a random number generator RAND.

**Method :** Initialize $\xi_i$ for $1 \leq i \leq W$ such that each class has only one object.

**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) ≠ 1) **Then** (*Move $O_i$ towards the internal state*)

$\xi_i = \xi_i - 1$

**EndIf**

**If** (($\xi_j$ mod N) ≠ 1) **Then** (*Move $O_j$ towards the the internal state*)

$\xi_j = \xi_j - 1$

**EndIf**

**Else** (* The partitioning is penalized *)

**If** ((($\xi_i$ mod N) ≠ 0) and (($\xi_j$ mod N) ≠ 0)) **Then**

(* Both are in internal states *)

$\xi_i = \xi_i + 1$

$\xi_j = \xi_j + 1$

**Else If** ($\xi_i$ mod N ≠ 0 **AND** $\xi_j$ mod N = 0 ) **Then**

$\xi_i = \xi_i + 1$ (* $O_i$ is in an internal state *)

**If** (RAND < δ) **Then** $\xi_j = \lceil \xi_i / N \rceil * N$

**Else If** ( $\xi_i$ mod N = 0 **AND** $\xi_j$ mod N ≠ 0) **Then**

$\xi_j = \xi_j + 1$ (* $O_j$ is in an internal state *)

**If** (RAND < δ) **Then** $\xi_i = \lceil \xi_j / N \rceil * N$

**Else** (* Both are in boundary states *)

temp1 = $\xi_i$

temp2 = $\xi_j$

**If** (RAND < δ) **Then** $\xi_i$ = temp2

**If** (RAND < δ ) **Then** $\xi_j$ = temp1

**EndIf**

**EndIf**

**EndFor**

Print out partitions based on the states $\xi_i$

**ForEver**

**End Algorithm SMA**

Examples which clarify the operation of the SMA are given elsewhere [32].

### IV.3 Experimental Results

The BAM was simulated to obtain experimental results so that comparisons can be made between it and the learning automata solutions presented in this paper. The objectives of the experiments were primarily to :

(a) obtain the rate of convergence and the accuracy as a function of the number of objects, when the objects are partitioned into classes of equal size.

(b) obtain the rate of convergence and the accuracy as a function of the number of objects, when objects are partitioned into classes, not necessarily of equal size.

Various values of W (number of objects) and R (number of classes) were used. In each case, **one hundred** experiments were conducted in order to obtain accurate results. Besides, each experiment was allowed to process a maximum of **50,000** queries.

The values of the $X_i$'s discussed in Section II were uniformly distributed in the interval [0,100] initially because a very tight initial distribution of $X_i$'s could lead to the covergence involving a single class. The range [0,100] used as a standard in all the experiments caused the BAM to yield good partitioning results, especially when the number of objects was large.

The three parameters required by the BAM, namely the $\Delta'_i$s and the neighbouring distance (used in identifying the potential classes) were assigned as follows :

(a) Since no *a priori* information was given, $\Delta_1$ and $\Delta_2$ were set to **unity** as suggested in [30,31].

(b) Although the $X_i$'s were initially distributed in the interval [0,100], they were spread throughout a fairly large interval after the BAM had been invoked for a while. In this case, if the distance between an object and its right hand side neighbour was set to a small value, then it would take quite a long time for the BAM to identify a reasonable set of potential classes. Therefore, we set this parameter to be 50, because, this value made the BAM identify a reasonable set of potential classes in a "reasonable" amount of time.

#### IV.3.1 Assumptions of the Underlying Groups

In all the experiments performed, queries were generated based on an underlying distribution unknown to the algorithms and were based on the following

assumptions :

(a) The probability of accessing any pair of objects in one group is the same as that of accessing any other pair of objects in the same group.

(b) The probability of accessing any pair of objects in different groups is the same as that of accessing any other pair of objects in different groups.

(c) The probability of accessing a pair of objects in the same group is greater than that of accessing a pair of objects in different groups.

Thus, if $\pi_i$ was the group in which the physical object $A_i$ is located, then these queries obeyed the following distribution:

$$\sum_{A_j \in \pi_i} \Pr[A_i, A_j \text{ accessed together}] = P, \quad i \neq j \tag{14}$$

Observe that if P = 1, then the partitioning is ideal (i.e., queries will involve only objects in the same partition). However, as the value of P decreases, the queries will be decreasingly informative about the underlying distribution governing the queries. Indeed, the value of P should be greater than 0.5 due to assumption (c).

The SMA discussed above was also simulated and compared with the BAM. The objectives were primarily to :

(a) compare the rate of convergence and the accuracy of both methods as a function of W, when the objects are partitioned into classes with different values of $\delta$ defined by (8), and,

(b) obtain the accuracy of both methods as a function of the joint access probability defined by (14), with different values of $\delta$.

In case (a), for each W (the number of objects), R (the number of classes, not necessarily of equal-size) and $\delta$, **one hundred** experiments were performed in order to obtain accurate results. In case (b), **one hundred** experiments were also performed for each set of values of P and $\delta$.

Initially, the objects were assigned arbitrarily to W distinct classes (actions) so that no two objects were assigned to the same class. The two parameters of the SMA, which are the number of states per action and $\delta$, were assigned as follows :

(a) Each action had 10 states associated with it.

(b) $\delta$ was set to 0.5 and 0.25 to see the effect of $\delta$ on the accuracy of the SMA.

As in [19], the SMA was considered to have converged to the optimal partitioning if all the objects moving around in the automaton were in the most

internal states of the actions chosen.

A summary of the results obtained is found in Tables 1-3. From Table 1 (a)–(b), we observe that the SMA converged much faster than the BAM. For example, when fifteen objects were partitioned into four unequal-size classes, the BAM converged after 3507 iterations. The corresponding figures for the SMA are 369 when $\delta = 0.25$ and 307 when $\delta = 0.5$. This implies an improvement by a factor of about 9.5 and 11.4 for $\delta = 0.25$ and 0.5, respectively. A comparative graphical sketch of the rates of convergence of the BAM and the SMA is drawn in Figure 2 as a function of the number of objects. In this case, for each point on the graph the number of classes is three.

Unfortunately however, the accuracy of the SMA is not as good as that of the BAM. As stated earlier, an experiment is said to have "failed" if the algorithm under consideration converged to a partition which is **different** from the underlying partitioning used in the simulation. From Table 2 (a)–(b), we observe that the BAM gave 100 per cent accuracy in all but two sets of experiments. In the case of the SMA, however, when $\delta = 0.5$, the accuracy ranges from 19 per cent to 88 per cent while the mean is 73.8 per cent. Further, when $\delta = 0.25$, the accuracy ranges from 33 per cent to 87 per cent while the mean is 74.8 per cent. Figure 3 illustrates the accuracy of the BAM and the SMA when $\delta = 0.5$ and 0.25 as a function of the number of objects. In this case, again, for each point on the graph, the number of classes is three.

The accuracy of the SMA and the BAM was also compared for various values of P, the joint access probability defined by (14), when six objects were partitioned into three equal-size classes. From Table 3 we observe that the accuracy of the SMA decreased as P decreased. For instance, when $P = 0.9$, the SMA gave 82 per cent accuracy for both $\delta = 0.25$ and 0.5. However, when $P = 0.75$, it only gave 47 per cent and 43 per cent accuracy for $\delta = 0.25$ and $\delta = 0.5$, respectively, and decreased to a rather poor 14 per cent and 5 per cent when $P = 0.6$. On the other hand, the accuracy of the BAM was **very stable** and maintained the value of 100 per cent even when P was decreasing. Figure 3 illustrates a comparative graphical sketch of the accuracy of the BAM and the SMA as a function of P. In this case, for each point on the graph the number of classes is three.

We conclude this section by stating that the SMA is to be preferred to the BAM if the user has *apriori information* that the underlying groups are "almost" of equal size.

# V. A VSSA APPROACH TO THE OPP

## V.1 Motivation

Although the SMA converges much faster than the BAM, in general, it does not always guarantee good accuracy. The inaccuracy of this solution cannot be blamed on the initialization process, because only one object is assigned to each action initially. However, it is clearly due to an imperfect way of moving objects around in the automaton, especially in the way the objects change actions. Consider the movements of the objects $O_i$ and $O_j$, which are in states $\xi_i$ and $\xi_j$, respectively. If neither $\xi_i$ nor $\xi_j$ is a boundary state, then no change of action will be required regardless of the response from the environment. On the other hand, if either $\xi_i$ or $\xi_j$ or both are in boundary states, then upon receiving a penalty from the environment, a change of action may result. But we know that the environment provides "misleading" information with probability 1–P. Although the objects are required to move only with probability $\delta$, the probability of moving incorrectly is still finite. Thus ultimately they still possess a high probability of moving incorrectly. We thus believe that we need a VSSA solution to the OPP.

In the process of searching for an improvement on the SMA, the objects moving around in the automaton were examined carefully with the hope that we could come up with a recovery scheme (i.e., a scheme which, when something went wrong, led back to the point where the process started going wrong). But before such a recovery scheme can be used, we must be able to first detect that a given situation is not correct. In order to do this, some properties of the objects must be identified in order to select those objects which lead to erroneous actions.

Let $O_i$, $O_j$ and $O_k$ be three distinct abstract objects in actions $\alpha_p$, $\alpha_q$ and $\alpha_r$, respectively and in states $\xi_i$, $\xi_j$ and $\xi_k$, respectively. Suppose the physical pair ($A_i$, $A_j$) is accessed more frequently than the pairs ($A_i$, $A_k$) and ($A_j$, $A_k$). Then, consider the situation when $O_i$ is in the boundary state of $\alpha_p$ (i.e., $\xi_i = \lceil \alpha_p / N \rceil * N$ assuming that there are N states associated with each action) while $O_j$ and $O_k$ are not in the boundary states of their actions. If the next query accesses the pair ($A_i$, $A_k$) and $O_i$ decides to move, it will move from the boundary state of $\alpha_p$ to the boundary state of $\alpha_r$, according to the transition map of the SMA. In other words, $\xi_i$ will be updated to have the value $\lceil \alpha_r / N \rceil * N$. Since $A_i$ and $A_k$ are seldom accessed together, $O_i$ is considered as having made a wrong move. As the queries are being processed, we observe that there are some special properties associated with $O_i$ which can be described as follows :

(1) Although $O_i$ is brought into $\alpha_r$ by $O_k$, the probability of having the

corresponding physical pair ($A_i$, $A_k$) accessed together is very small.

(2) Although there may be other objects in $\alpha_r$, most of them (if not all of them) are seldom accessed together with $O_i$. In other words, ideally $O_i$ will not have any "partners" in $\alpha_r$.

(3) Whenever $O_i$ is accessed together with another object, say $O_j$, both of them will probably be penalized, because there is a high probability that $O_j$ will not be in $\alpha_r$. Similarly, receiving a reward is unlikely.

The first property does not seem very useful unless we have frequency counters on all pairs of accessed objects. However, the use of such counters is discouraged for it defeats the purpose of learning. The second property is not useful either, for it does not seem possible to determine whether an object is in a correct action based on the number of "partners" it has in the same action. However, the last property does provide some useful information because frequent penalties received by $O_i$ indicates that $O_i$ may be in a wrong action. Thus a departure from that action may become necessary (each penalty brings $O_i$ closer to the boundary state of the current action that it chooses). If an object accessed together with $O_i$ (say $O_j$), did not move accidentally into the action currently chosen by $O_i$ (say $\alpha_p$), then $O_i$ would be able to move out of $\alpha_p$. However, if $O_j$ reached the boundary state first and decided to move to $\alpha_p$, it would be much harder for **both** of them to leave $\alpha_p$.

In fact, underlying this property is an important concept which we informally state: "the longer $O_i$ stays in an erroneous action $\alpha_p$, the more trouble it causes". Thus, if $O_i$ accidentally moves into a wrong action but is able to leave that action right away, not only will it be able to recover from the wrong transition, but it will also prevent other objects from entering that action when they are accessed together with $O_i$. The understanding of this important property motivates the use of **variable structure** stochastic automata, primarily because these automata allow objects to choose **two different** actions at **two consecutive** time instants. Thus, even if an object accidentally chooses a wrong action at the current instant, it is possible for it to choose another action at the next instant.

In the following sections, we shall propose a new VSSA called the Modified Linear Reward-Penalty ($ML_{RP}$) automaton. This automaton employs a linear updating scheme similar to the Linear Reward-Penalty ($L_{RP}$) scheme. Due to the similarity between the $L_{RP}$ scheme and the one used in the $ML_{RP}$ automaton, we shall first briefly review the $L_{RP}$ scheme.

## V.2 The Linear Reward-Penalty Scheme

As described in [10], the Linear Reward-Penalty ($L_{RP}$) scheme is the earliest known updating scheme. It is a linear scheme because the $\mathbf{p}(n+1)$ term is a linear function of the components of $\mathbf{p}(n)$. The $L_{RP}$ scheme is defined as :

$$
\begin{aligned}
p_i(n+1) &= p_i(n) + \textstyle\sum_{j\neq i} a * p_j(n) \\
&= p_i(n) + a * (1 - p_i(n)) && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\
p_i(n+1) &= p_i(n) - a * p_i(n) \\
&= (1 - a) * p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 0 \\
p_i(n+1) &= p_i(n) - \textstyle\sum_{j\neq i} \{[b/(R-1)] - b * p_j(n)\} \\
&= p_i * (1 - b) && \text{if } \alpha_i \text{ is chosen and } \beta = 1 \\
p_i(n+1) &= p_i(n) + [b/(R-1)] - b * p_i(n) \\
&= [b/(R-1)] + (1 - b) * p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 1
\end{aligned}
$$

If there are R actions in the action set, then $1 \leq i, j \leq R$.

The $L_{RP}$ scheme has been extensively studied in the literature. It has been shown that the scheme is ergodic and that an automaton using the $L_{RP}$ scheme is expedient in all stationary random environments.

## V.3 The Modified Linear Reward-Penalty Scheme

Just like the OMA and the SMA, the objects to be partitioned are represented by a set of abstract objects which are allowed to move around in the automaton. The rationale for the $ML_{RP}$ scheme is as follows: When a pair of objects $(A_i, A_j)$ is accessed if $O_i$ and $O_j$ choose the same action, say $\alpha_k$, then they will be rewarded by the interacting environment. Just like the $L_{RP}$ scheme, this is achieved by **increasing** the probability of them choosing $\alpha_k$ and at the same time **decreasing** the probabilities of them choosing **all other** actions. On the other hand, if $O_i$ and $O_j$ choose actions $\alpha_p$ and $\alpha_q$ respectively, where $\alpha_p \neq \alpha_q$, then they will be penalized. Since it **may be** the case that $A_i$ and $A_j$ should be in the same class, both $O_i$ and $O_j$ should "try" to use a common action. In such a case, instead of letting $O_i$ move to $\alpha_q$ and $O_j$ move to $\alpha_p$, we modify the probabilities so that $O_i$ chooses $\alpha_q$ and $O_j$ chooses $\alpha_p$ with higher probabilities. Since the other actions $\alpha_k$ (for $k \neq i, j$) are not involved, the probabilities of them being chosen by $O_i$ and $O_j$ should not be changed. To be specific, since $\alpha_p$ is considered as an inappropriate action for $O_i$ and $\alpha_q$ is the one that $O_i$ should try, $p_{ip}$ (the probability of $O_i$ choosing

$\alpha_p$) should be decremented. Consequently, the decrement should be added to $p_{iq}$ (the probability of $O_i$ choosing $\alpha_q$). Similarly, the value of $p_{jq}$ should be decreased and the value of $p_{jp}$ should be increased by the corresponding amount. However, since we are not sure whether the accessed pair ($A_i$, $A_j$) should really be placed in the same class, the amount of change upon receiving a penalty should be **small** so that a recovery due to an erroneous choice can be more easily achieved. We shall now formally define the $ML_{RP}$ automaton.

If $\{A_1, ..., A_W\}$ is the set of W given objects, then just like the SMA, there will be W actions in the $ML_{RP}$ automaton with each action representing a certain class. The $ML_{RP}$ is defined as a quadruple ($\alpha$, $\beta$, P, F) where

(i) $\alpha = \{\alpha_1, ..., \alpha_W\}$ is the set of actions that the abstract objects must choose from.

(ii) $\beta = \{0, 1\}$ is the set of inputs. The value '0' represents a reward, '1' is a penalty.

(iii) $P = \{P_1, ..., P_W\}$ is the set of action probability vectors such that $P_i = [p_{i1},...,p_{iW}]^T$, for all $1 \le i \le W$. Each component $p_{ij}$, $1 \le j \le W$, represents the probability that action $\alpha_j$ will be chosen by $O_i$. Also, $\Sigma_{1 \le j \le W}\ p_{ij} = 1.0$

(iv) F is the probability updating scheme. If $A_i$ and $A_j$ are the accessed objects and their abstract counterparts choose actions $\alpha_m$ and $\alpha_n$, respectively, $1 \le m, n \le W$, then the $ML_{RP}$ scheme is defined as follows, for $0 < \lambda_1, \lambda_2 < 1$.

$$\begin{cases} p_{im}(n+1) = 1 - \lambda_1 * (1 - p_{im}(n)) \\ p_{ik}(n+1) = \lambda_1 * p_{ik}(n) \end{cases} \quad \text{if } \beta = 0 \text{ (i.e., } \alpha_m = \alpha_n) \quad 1 \le k \le W \text{ and } k \ne i$$

$$\begin{cases} p_{in}(n+1) = p_{in}(n) + (1 - \lambda_2) * p_{im}(n) \\ p_{im}(n+1) = \lambda_2 * p_{im}(n) \end{cases} \quad \text{if } \beta = 1 \text{ (i.e., } \alpha_m \ne \alpha_n)$$

The probabilities $\{p_{jk}\}$ are updated in an analogous manner.

Just like the SMA discussed earlier, the "query-system" is the environment which the $ML_{RP}$ automaton interacts with. The input to it is the set of actions chosen by the abstract objects. Every time a pair of objects ($A_i$, $A_j$) is accessed, the environment compares the actions that are chosen by the abstract objects, say $\alpha_p$ for $O_i$ and $\alpha_q$ for $O_j$, and sends out a response according to the following rules :

(1) If $\alpha_p = \alpha_q$, then a reward (i.e., $\beta = 0$), will be sent out.

(2) If $\alpha_p \ne \alpha_q$, then a penalty (i.e., $\beta = 1$), will be sent out.

## V.4 Implementation Details

If $\{O_1, ..., O_W\}$ is the set of W abstract objects, then there will be W actions $\{\alpha_1, ..., \alpha_W\}$ available for each of these objects. Since each object can be in any of the W actions, there is an action probability vector $[p_{i1}, ..., p_{iW}]^T$ associated with object $O_i$, for $1 \leq i \leq W$, such that the value $p_{ik}$ represents the probability that $O_i$ will choose action $\alpha_k$ at a given instant. Just like the SMA, each object $O_i$ is initially assigned to choose an action $\alpha_i$. Thus, if $O_i$ and $O_j$ are two distinct objects, then they will not be initially assigned to the same action. With no loss of generality, we can assume that $O_i$ is assigned to choose $\alpha_i$ (i.e., $p_{ii} = 1.0$ and $p_{ij} = 0.0$ for $1 \leq i, j \leq W$ and $i \neq j$).

When a pair of objects $(A_i, A_j)$ is accessed, if the corresponding abstract objects choose actions $\alpha_p$ and $\alpha_q$, respectively, then they will be rewarded if $\alpha_p = \alpha_q$; otherwise, they will be penalized. Upon receiving a reward, both the values of $p_{ip}$ and $p_{jq}$ will be increased while all other components of the action probability vectors associated with $O_i$ and $O_j$ will be correspondingly decreased according to the $ML_{RP}$ scheme. In the case of receiving a penalty, the values of $p_{iq}$ and $p_{jp}$ will be increased and **only** $p_{ip}$ and $p_{jq}$ will be decreased.

After T accesses, the physical objects will be partitioned based on the actions chosen by the abstract objects. For the sake of completeness, we present below an algorithmic version of the $ML_{RP}$ automaton.

### Algorithm $ML_{RP}$ Automaton

**Input** : The abstract objects $\{O_1, ..., O_W\}$, the reward parameter $\lambda_1$ and the penalty parameter $\lambda_2$. Also coming into the algorithm is a stream of queries $(A_i, A_j)$.

**Output** : A periodic partitioning of the objects.

**Notation** : $\alpha_p$ and $\alpha_q$ are the actions chosen by $O_i$ and $O_j$ at a given instant. They are integers between 1 and W. $p_{ij}$ is the probability that $O_i$ chooses group j at a given instant, $0 \leq p_{ij} \leq 1.0$, $\Sigma_{1 \leq j \leq W}\ p_{ij} = 1.0$

**Method** : With no loss of generality, initialize $p_{ii} = 1.0$ and $p_{ij} = 0.0$.

**Repeat**

**For** a sequence of T queries **do**

ReadQuery ($A_i$, $A_j$)

**If** ($\alpha_p = m$ and $\alpha_q = m$) (* Reward *)

$p_{im} = 1 - \lambda_1 ( 1 - p_{im})$

$p_{jm} = 1 - \lambda_1 ( 1 - p_{jm})$

**For** ( k = 1 to W, $k \neq m$)

$p_{ik} = \lambda_1 p_{ik}$

$p_{jk} = \lambda_1 p_{jk}$

**EndFor**

**EndIf**

**If** ( $\alpha_p = m$ and $\alpha_q = n$ ) (* Penalty *)

$p_{in} = p_{in} + (1 - \lambda_2) p_{im}$

$p_{im} = \lambda_2 p_{im}$

$p_{jm} = p_{jm} + (1 - \lambda_2) p_{jn}$

$p_{jn} = \lambda_2 p_{jn}$

**EndIf**

**EndFor**

Print out partitions based on the probabilities $p_{ij}$

**ForEver**

**End Algorithm $ML_{RP}$**

## V.5 Experimental Comparison with the BAM

The $ML_{RP}$ discussed above was simulated and compared with the BAM primarily to:

(a) compare the rate of convergence and the accuracy of the $ML_{RP}$ automaton with the BAM as a function of the number of objects to be partitioned,

(b) compare the rate of convergence and the accuracy of the $ML_{RP}$ automaton with the BAM as a function of the joint access probability defined by (14).

In each set up, **one hundred** experiments were performed in order to obtain accurate results. Regarding initialization, if W objects were given, then initially each object $O_i$ was assigned arbitrarily to an action $\alpha_j$, for $1 \leq i, j \leq W$, with probability 1 (i.e., $p_{ij} = 1.0$ and $p_{ik} = 0.0$ for all $1 \leq k \leq W$ and $k \neq j$).

The two parameters determining the $ML_{RP}$ scheme, namely, $\lambda_1$ and $\lambda_2$, and the probability P defined by (14) were assigned as follows:

(1) $\lambda_1$ and $\lambda_2$ were set to 0.5 and 0.975 respectively in all the experiments performed.

(2) when the solutions were compared as a function of the number of objects, the number of objects varied from four to eighteen. In the case when the joint access probability P defined by (14) was varied, the number of objects was six and the number of classes was three. In the latter case, P was chosen from the interval [0.6, 0.9].

The parameters and assumptions used in the case of the BAM were the same as those described in Section IV. The $ML_{RP}$ automaton was considered to have converged if every object $O_i$, $1 \leq i \leq W$, moving around in the automaton chose any of the W actions with a probability of 0.95 or more. This is quite realistic because when such a situation occurs the object will not be able to change to other actions easily.

A summary of the results that have been obtained is found in Tables 3 and 4. From Table 3, we observe the superiority of the rate of convergence of the $ML_{RP}$ automaton to the BAM. For example, when fifteen objects were partitioned into four classes of unequal size, the BAM converged after 3507 iterations. The corresponding figure for the $ML_{RP}$ automaton when $\lambda_1 = 0.5$ and $\lambda_2 = 0.975$ is 383. This is an improvement by a factor of about 9.1. A comparative graphical sketch of the average number of iterations required for convergence of the BAM and the $ML_{RP}$ automaton is drawn in Figure 4 where objects were partitioned into three classes of equal size.

In terms of accuracy, we observe from Table 4 that the BAM gave excellent accuracy, namely between 81 per cent and 100 per cent with the mean of 99.1 per cent. In the case of the $ML_{RP}$ automaton, when $\lambda_1 = 0.5$ and $\lambda_2 = 0.975$, the accuracy ranged from 95 per cent to 100 per cent with the mean of 98.6 per cent. This implies that the BAM is about 0.5 per cent more accurate than the $ML_{RP}$ automaton. Figure 5 compares the accuracy of the BAM and the $ML_{RP}$ automaton graphically as a function of the number of objects. In this case, for each point on the graph the number of classes is three.

Table 5 summarizes the performance of the BAM and the $ML_{RP}$ automaton in various environments (i.e., for various joint access probabilities P defined by (14)). From the table, we observe that, in general, the BAM converged more slowly than the $ML_{RP}$ automaton. For example, when P = 0.8, the BAM converged after 332

iterations and the $ML_{RP}$ automaton converged after 72 iterations. The $ML_{RP}$ automaton demonstrated an improvement by a factor of 4.6. In terms of accuracy, we observe that the BAM provided excellent and stable accuracy (i.e., 100 per cent in the various environments). In the case of the $ML_{RP}$ automaton, when it interacted with an environment characterized by the parameter P, in general, it yielded 100 per cent accuracy with probability at least P. For example, when P = 0.65, the $ML_{RP}$ automaton gave 77 per cent accuracy. Figure 6 graphically illustrates the comparison of the accuracy of the BAM and the $ML_{RP}$ automaton as a function of the joint access probability P.

## V.6 Comments

The $ML_{RP}$ scheme discussed in this chapter shows great improvement on accuracy over the SMA. Although it appears to be marginally less accurate than the BAM, the accuracy obtained by this scheme is definitely acceptable. Typically, a 95 per cent accuracy is obtained in an environment characterized by the parameter P = 0.9. However, in all cases, the $ML_{RP}$ scheme is **much faster** than the BAM.

Experimental results suggest that $\lambda_1$ should take a value in the neighbourhood of 0.5 and $\lambda_2$ should take a value close to unity, $0 < \lambda_1, \lambda_2 < 1$. In the case of $\lambda_2$, using a larger value (i.e., closer to unity) is quite plausible intuitively because, this implies that on being penalized only small changes are made on the action probabilities. Therefore, any erroneous choice can be rectified easily. On the other hand, the fact that $\lambda_1$ should take on values near 0.5 seems to contradict our intuition because this causes substantial changes on the action probabilities upon receiving rewards. This is intuitively not very desirable since the information leading to a reward may be misleading. We are unable to explain this counter-intuitive behaviour.

Although the $ML_{RP}$ scheme provides such good results, analyzing its $\varepsilon$-optimality properties are definitely non-trivial. We believe that this involves analyzing a complex co-operative automaton game between the participating abstract objects. However, in a straightforward way, the $ML_{RP}$ automaton can be shown to be **expedient,** and experimental results seem to indicate that it is $\varepsilon$-optimal as well. We shall show that it is expedient in the following theorem.

**Theorem V**

The $ML_{RP}$ automaton is expedient if $0 < \lambda_1, \lambda_2 < 1.0$.

**Proof :**

Each object $O_i$ moving around in the automaton is initially assigned to a distinct action (representing a class) $\alpha_k$, $1 \leq i, k \leq W$, $P_{ik} = 1.0$ and $P_{ij} = 0.0$, for all $1 \leq j \leq W$ and $j \neq k$. With no loss of generality, we assume $i = k$. Since each $O_i$ is in a distinct class, the probability of an accessed pair of objects being rewarded is zero. Thus, the initial expected penalty, denoted as $M_0$, is **unity**.

Suppose at the first time instant the pair $(A_i, A_j)$ is accessed. Since $P_{ii} = 1.0$ and $P_{jj} = 1.0$, $O_i$ and $O_j$ will not be able to choose the same class. Thus both $O_i$ and $O_j$ will be penalized. According to the $ML_{RP}$ scheme, $P_{ii}$ is updated to $\lambda_2 * P_{ii}$ and $P_{ij}$ is updated to $P_{ij} + (1 - \lambda_2) * P_{ii}$. Similarly $P_{jj}$ changes to $\lambda_2 * P_{jj}$ and $P_{ji}$ changes to $P_{ji} + (1 - \lambda_2) * P_{jj}$. After the first query, the expected probability that $O_i$ and $O_j$ will be rewarded if $(A_i, A_j)$ is accessed will become **non-zero**, if $0.0 < \lambda_2 < 1.0$. Thus, if we denote the expected penalty at the first time instant as $E[M(1)]$, then $E[M(1)] < M_0$.

After the first instant, independent of whether $O_i$ and $O_j$ are rewarded or penalized, the values of both $P_{ii}$ and $P_{jj}$ cannot be changed back to 1 if $0 < \lambda_1, \lambda_2 < 1$. Similarly, the values of both $P_{ij}$ and $P_{ji}$ cannot be changed back to 0. Therefore, the limiting expected penalty, $E[M(\infty)]$, will be less than $M_0$, and hence the theorem is proved. ♦ ♦ ♦

## VI. CONCLUSIONS

In this paper, we have studied the Object Partitioning Problem (OPP). This problem involves partitioning a set of objects $\{A_1, A_2, ..., A_W\}$ into classes whose sizes are not necessarily equal. The intention is to partition the set such that the objects accessed together are found in the same class.

In an earlier paper, we studied a special case of the OPP, namely the case in which the objects were to be **equi-partitioned**. The solution which we proposed involved a new learning automaton, which was called the Object Migrating Automaton (OMA). In this paper, we have first presented a stochastic version of the OMA as a solution to the OPP. The latter, which is called the Stochastic Move Automaton (SMA) is much faster than the BAM, the best known technique known in the literature. However, its accuracy is not too remarkable in certain environments.

Additionally, we have presented a variable structure stochastic automaton

solution for the OPP. This involves a new automaton closely related to the traditional Linear Reward-Penalty Scheme. The new automaton, the $ML_{RP}$ automaton, is almost as accurate as the BAM, but converges an order of magnitude faster. In certain environments, the latter converges about 20 times faster than the BAM. We have shown in a rather straight forward way that the $ML_{RP}$ automaton is expedient, and we conjecture that it is $\varepsilon$-optimal in all random environments.

By way of comparison, we recommend that the SMA be used for the partitioning process if the user has *a priori* information that the underlying groups are "almost" of the same size. In the absence of any *a priori* information we suggest that the $ML_{RP}$ automaton be employed to achieve the partitioning.

**Acknowledgements :** We are grateful to David Ng who helped prepare the final manuscript. We are also very grateful to Dr. Reichstein who painstakingly proofread the final manuscript to render it more easily readable.

## 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., and Ma, D.C.Y, "Deterministic Learning Automata Solutions to the Equi-Partitioning Problem". To appear in the IEEE Trans. on Computers.

20. Oommen, B.J., and Hansen, E.R.,"List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations", Proc. of the 1986 International Conference on Database Theory, Rome, Italy, September 1986.

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., Suen, C.M., Lam, K. and Siu, M.K.,, "Adaptive Record Clustering ", ACM Trans. on Database Systems, 1985, pp.180-204.

31. 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.

32. Ma, D.C.Y., "Object Partitioning by using Learning Automata", M.C.S. Thesis, School of Computer Science, Carleton University, Ottawa, Canada, April 1986.

| No. Objects | Objects/Class | No. Classes | # iterations BAM | # iterations SMA |
|---|---|---|---|---|
| 4 | 2 | 2 | (21, 40) | (28, 54) |
| 6 | 2 | 3 | (255, 269) | (50, 85) |
| | 2, 4 | 2 | (55, 70) | (85, 137) |
| | 3 | 2 | (243, 251) | (62, 98) |
| 9 | 3 | 3 | (745, 762) | (91, 156) |
| | 2, 3, 4 | 3 | (365, 377) | (136, 239) |
| | 4, 5 | 2 | (711, 728) | (128, 202) |
| 12 | 2 | 6 | (1520, 1527) | (98, 187) |
| | 2, 2, 2, 3, 3 | 5 | (1940, 1977) | (154, 290) |
| | 3 | 4 | (1885, 1894) | (138, 231) |
| | 2,3, 3, 4 | 4 | (8193, 8197) | (192, 332) |
| | 4 | 3 | (2914, 2925) | (165, 265) |
| | 6 | 2 | (3593, 3609) | (201, 290) |
| 15 | 3 | 5 | (2501, 2508) | (182, 305) |
| | 2, 3, 3, 3, 4 | 5 | (8224, 8281) | (259, 430) |
| | 3, 4, 4, 4 | 4 | (3380, 3507) | (233, 369) |
| | 5 | 3 | (6442, 6445) | (251, 374) |
| 18 | 2 | 9 | (3348, 3352) | (186, 368) |
| | 3 | 6 | (2910, 2917) | (230, 385) |
| | 6 | 3 | (12754, 12756) | (331, 474) |
| | 9 | 2 | Not Available | (406, 543) |

**Table 1 (a)** Comparison of the rate of convergence of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P, the probability defined by (14), is 0.9 and $\delta$, the probability defined by (8), is 0.25

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

| No. Objects | Objects/Class | No. Classes | # iterations BAM | # iterations SMA |
|---|---|---|---|---|
| 4 | 2 | 2 | (21, 40) | (25, 51) |
| 6 | 2 | 3 | (255, 269) | (46, 79) |
| | 2, 4 | 2 | (55, 70) | (70, 115) |
| | 3 | 2 | (243, 251) | (50, 84) |
| 9 | 3 | 3 | (745, 762) | (78, 138) |
| | 2, 3, 4 | 3 | (365, 377) | (105, 206) |
| | 4, 5 | 2 | (711, 728) | (101, 175) |
| 12 | 2 | 6 | (1520, 1527) | (87, 172) |
| | 2, 2, 2, 3, 3 | 5 | (1940, 1977) | (116, 247) |
| | 3 | 4 | (1885, 1894) | (103, 192) |
| | 2,3, 3, 4 | 4 | (8193, 8197) | (139, 284) |
| | 4 | 3 | (2914, 2925) | (133, 230) |
| | 6 | 2 | (3593, 3609) | (158, 255) |
| 15 | 3 | 5 | (2501, 2508) | (143, 272) |
| | 2, 3, 3, 3, 4 | 5 | (8224, 8281) | (185, 363) |
| | 3, 4, 4, 4 | 4 | (3380, 3507) | (166, 307) |
| | 5 | 3 | (6442, 6445) | (187, 318) |
| 18 | 2 | 9 | (3348, 3352) | (169, 257) |
| | 3 | 6 | (2910, 2917) | (175, 339) |
| | 6 | 3 | (12754, 12756) | (255, 399) |
| | 9 | 2 | Not Available | (277, 412) |

**Table 1 (b)** Comparison of the rate of convergence of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P, the probability defined by (14), is 0.9 and $\delta$, the probability defined by (8), is 0.5

**Notation :** Same as Table 1(a).

| No. Objects | Objects/Class | No. Classes | Accuracy BAM | Accuracy SMA |
|---|---|---|---|---|
| 4 | 2 | 2 | 100% | 87% |
| 6 | 2 | 3 | 100% | 82% |
| | 2, 4 | 2 | 100% | 84% |
| | 3 | 2 | 100% | 84% |
| 9 | 3 | 3 | 100% | 80% |
| | 2, 3, 4 | 3 | 100% | 78% |
| | 4, 5 | 2 | 100% | 82% |
| 12 | 2 | 6 | 100% | 63% |
| | 2, 2, 2, 3, 3 | 5 | 100% | 73% |
| | 3 | 4 | 100% | 78% |
| | 2,3, 3, 4 | 4 | 96% | 79% |
| | 4 | 3 | 100% | 76% |
| | 6 | 2 | 81% | 80% |
| 15 | 3 | 5 | 100% | 77% |
| | 2, 3, 3, 3, 4 | 5 | 100% | 72% |
| | 3, 4, 4, 4 | 4 | 100% | 76% |
| | 5 | 3 | 100% | 80% |
| 18 | 2 | 9 | 100% | 33% |
| | 3 | 6 | 100% | 60% |
| | 6 | 3 | 100% | 67% |
| | 9 | 2 | Not Available | 79% |

**Table 2 (a)** Comparison of the accuracy of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P, the probability defined by (14), is 0.9 and $\delta$, the probability defined by (8), is 0.25

| No. Objects | Objects/Class | No. Classes | Accuracy BAM | Accuracy SMA |
|---|---|---|---|---|
| 4 | 2 | 2 | 100% | 87% |
| 6 | 2 | 3 | 100% | 82% |
| | 2, 4 | 2 | 100% | 83% |
| | 3 | 2 | 100% | 84% |
| 9 | 3 | 3 | 100% | 77% |
| | 2, 3, 4 | 3 | 100% | 73% |
| | 4, 5 | 2 | 100% | 88% |
| 12 | 2 | 6 | 100% | 53% |
| | 2, 2, 2, 3, 3 | 5 | 100% | 67% |
| | 3 | 4 | 100% | 80% |
| | 2,3, 3, 4 | 4 | 96% | 80% |
| | 4 | 3 | 100% | 78% |
| | 6 | 2 | 81% | 83% |
| 15 | 3 | 5 | 100% | 69% |
| | 2, 3, 3, 3, 4 | 5 | 100% | 71% |
| | 3, 4, 4, 4 | 4 | 100% | 72% |
| | 5 | 3 | 100% | 87% |
| 18 | 2 | 9 | 100% | 19% |
| | 3 | 6 | 100% | 68% |
| | 6 | 3 | 100% | 75% |
| | 9 | 2 | Not Available | 75% |

**Table 2 (b)** Comparison of the accuracy of the BAM and the SMA solutions for the OPP. In the latter case, the number of states per action is 10. P, the probability defined by (14), is 0.9 and $\delta$, the probability defined by (8), is 0.5.

| No. Objects | Objects/Class | No. Classes | # iterations BAM | # iterations $ML_{RP}$ |
|---|---|---|---|---|
| 4 | 2 | 2 | (21, 40) | (34, 37) |
| 6 | 2 | 3 | (255, 269) | (56, 60) |
| | 2, 4 | 2 | (55, 70) | (157, 163) |
| | 3 | 2 | (243, 251) | (91, 95) |
| 9 | 3 | 3 | (745, 762) | (144, 151) |
| | 2, 3, 4 | 3 | (365, 377) | (249, 258) |
| | 4, 5 | 2 | (711, 728) | (243, 250) |
| 12 | 2 | 6 | (1520, 1527) | (120, 130) |
| | 2, 2, 2, 3, 3 | 5 | (1940, 1977) | (242, 252) |
| | 3 | 4 | (1885, 1894) | (212, 222) |
| | 2,3, 3, 4 | 4 | (8193, 8197) | (326, 342) |
| | 4 | 3 | (2914, 2925) | (279, 288) |
| | 6 | 2 | (3593, 3609) | (382, 392) |
| 15 | 3 | 5 | (2501, 2508) | (279, 291) |
| | 2, 3, 3, 3, 4 | 5 | (8224, 8281) | (403, 416) |
| | 3, 4, 4, 4 | 4 | (3380, 3507) | (371, 383) |
| | 5 | 3 | (6442, 6445) | (467, 478) |
| 18 | 2 | 9 | (3348, 3352) | (216, 233) |
| | 3 | 6 | (2910, 2917) | (322, 335) |
| | 6 | 3 | (12754, 12756) | (606, 619) |
| | 9 | 2 | Not Available | (797, 813) |

**Table 3** Comparison of the rate of convergence of the BAM and the $ML_{RP}$ automaton solutions for the OPP. In the latter case, the values of $\lambda_1$ and $\lambda_2$ are 0.5 and 0.975, respectively. Also, the joint access probability P defined by (14) is 0.9.

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

| No. Objects | Objects/Class | No. Classes | Accuracy BAM | Accuracy $ML_{RP}$ |
|---|---|---|---|---|
| 4 | 2 | 2 | 100% | 99% |
| 6 | 2 | 3 | 100% | 100% |
| | 2, 4 | 2 | 100% | 95% |
| | 3 | 2 | 100% | 99% |
| 9 | 3 | 3 | 100% | 100% |
| | 2, 3, 4 | 3 | 100% | 99% |
| | 4, 5 | 2 | 100% | 96% |
| 12 | 2 | 6 | 100% | 99% |
| | 2, 2, 2, 3, 3 | 5 | 100% | 100% |
| | 3 | 4 | 100% | 99% |
| | 2,3, 3, 4 | 4 | 96% | 100% |
| | 4 | 3 | 100% | 100% |
| | 6 | 2 | 81% | 99% |
| 15 | 3 | 5 | 100% | 100% |
| | 2, 3, 3, 3, 4 | 5 | 100% | 99% |
| | 3, 4, 4, 4 | 4 | 100% | 100% |
| | 5 | 3 | 100% | 99% |
| 18 | 2 | 9 | 100% | 100% |
| | 3 | 6 | 100% | 100% |
| | 6 | 3 | 100% | 100% |
| | 9 | 2 | Not Available | 99% |

**Table 4** Comparison of the accuracy of the BAM and the $ML_{RP}$ automaton solutions for the OPP. In the latter case, the values of $\lambda_1$ and $\lambda_2$ are 0.5 and 0.975, respectively. Also, the joint access probability P defined by (14) is 0.9.

| P | # iterations BAM | # iterations $ML_{RP}$ | Accuracy BAM | Accuracy $ML_{RP}$ |
|---|---|---|---|---|
| 0.9 | (255, 269) | (56, 60) | 100% | 100% |
| 0.85 | (272, 285) | (60, 65) | 100% | 98% |
| 0.8 | (322, 332) | (67, 72) | 100% | 98% |
| 0.75 | (341, 349) | (71, 78) | 100% | 89% |
| 0.7 | (375, 385) | (88, 95) | 100% | 85% |
| 0.65 | (414, 421) | (94, 102) | 100% | 77% |
| 0.6 | (493, 501) | (123, 131) | 100 % | 54% |

**Table 5** Comparison of the converging rate and the accuracy of the BAM and the $ML_{RP}$ automaton as a function of the joint access probability defined by (14) when six objects are partitioned into three equal-size classes.

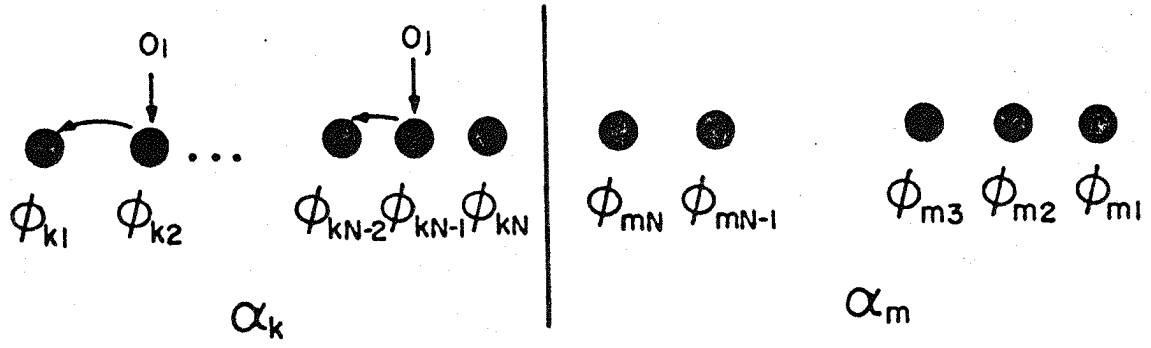


FIGURE I (a)

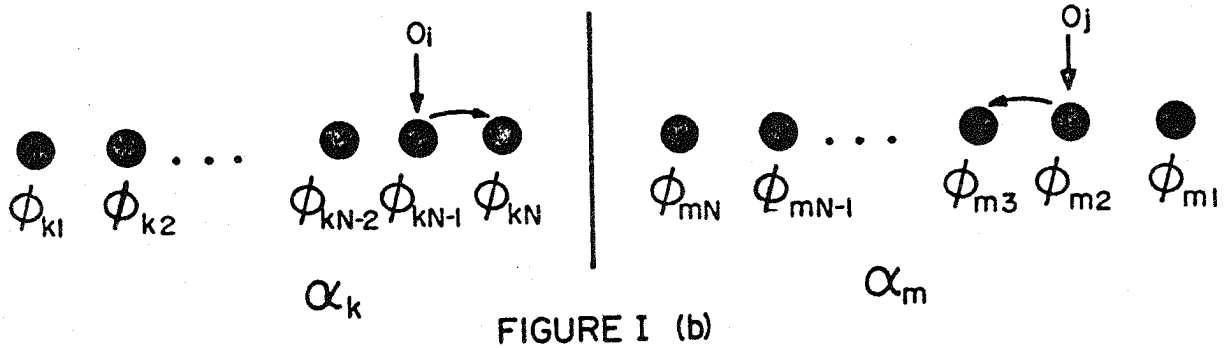


FIGURE I (b)

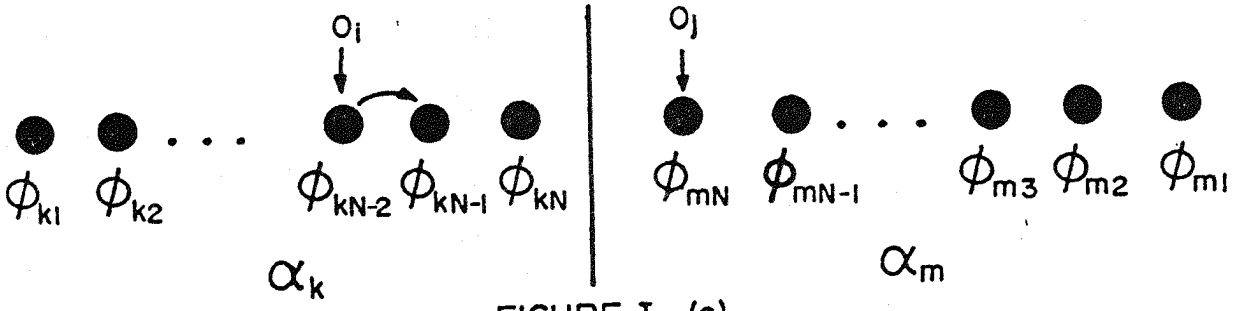


FIGURE I (c)

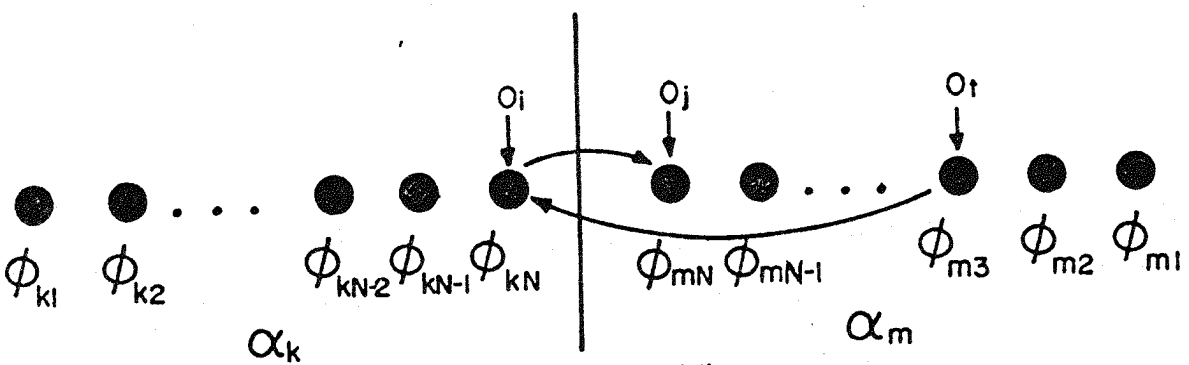


FIGURE I (d)

FIGURE I (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).

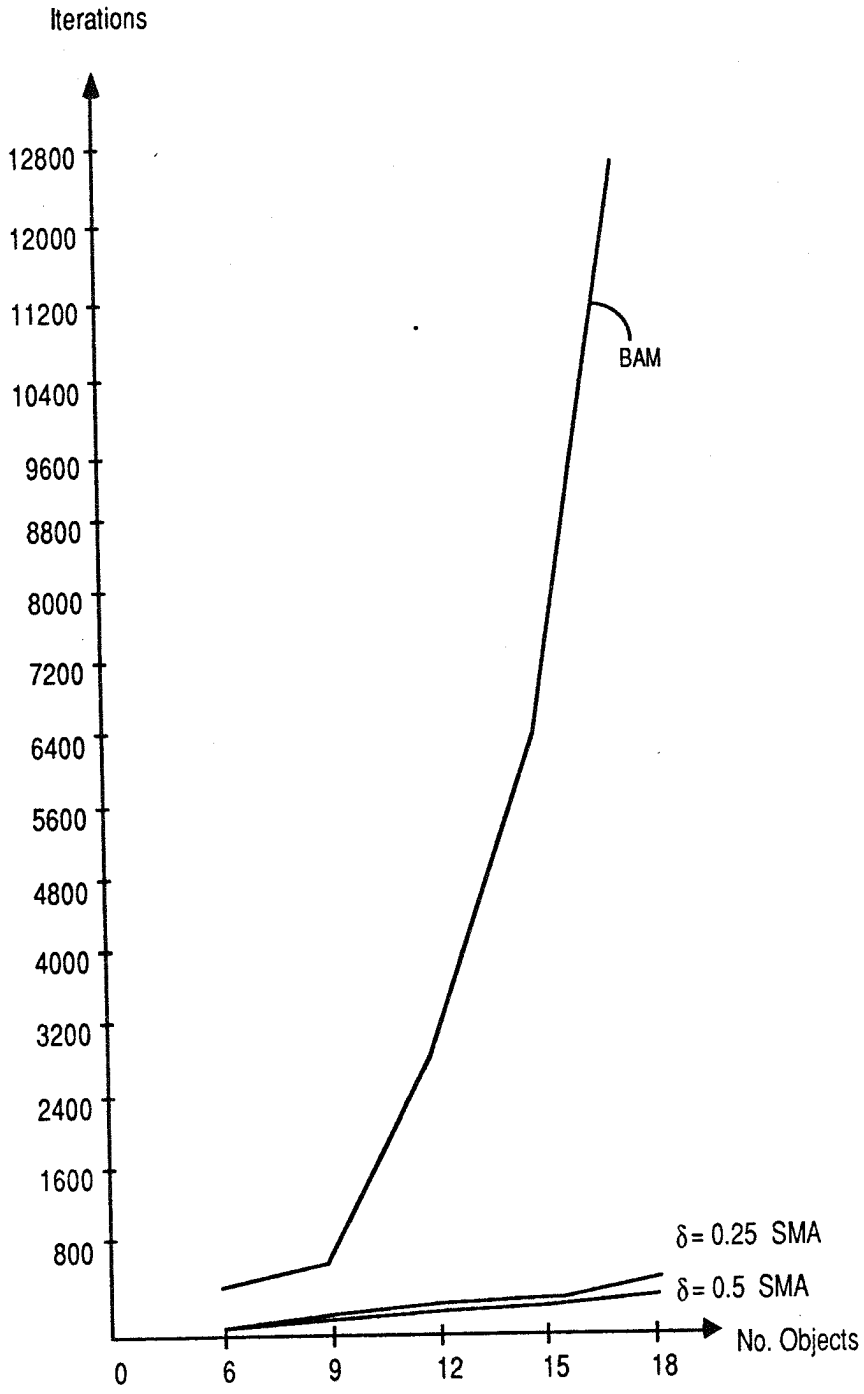


**Figure 2** Plot of the average number of iterations required for convergence for the BAM and the SMA as the number of objects increases. In each case the number of classes is three.

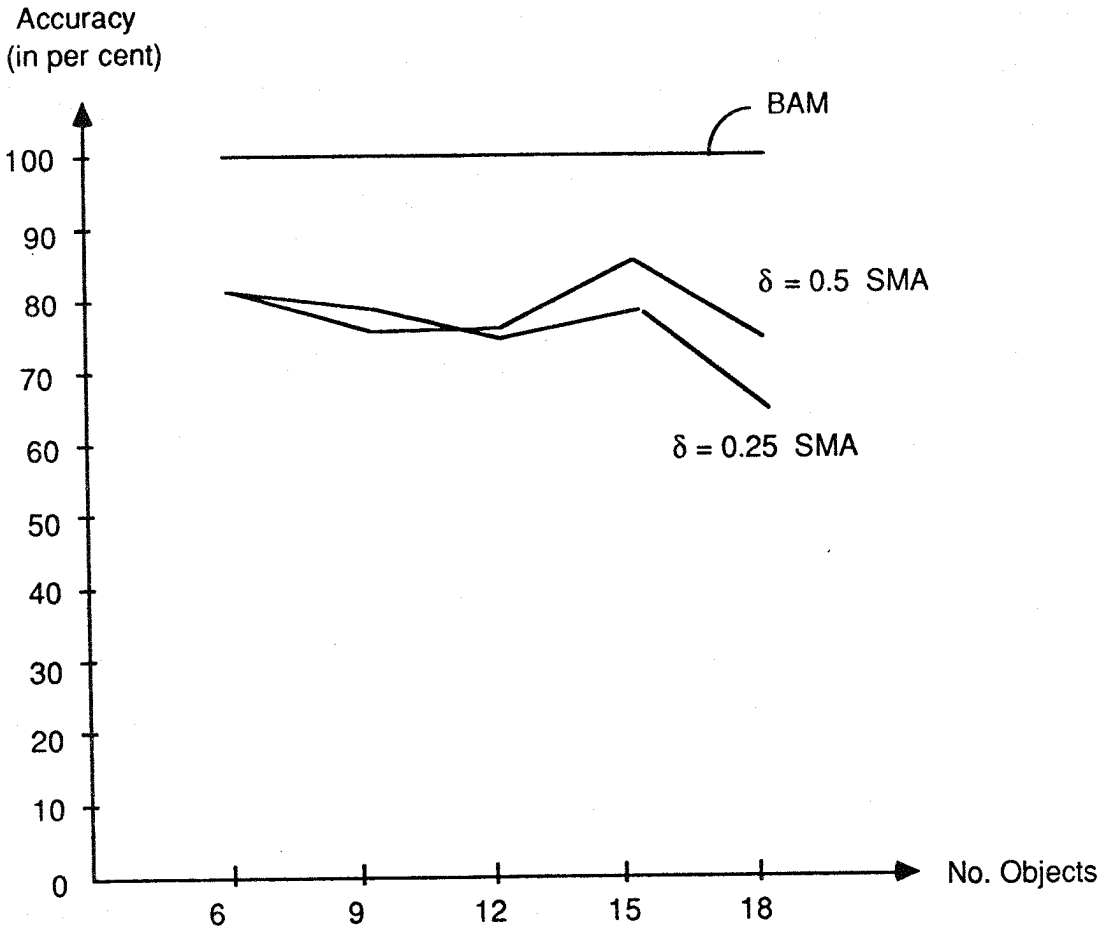


**Figure 3** Plot of the accuracy of the BAM and the SMA as a function of the number of objects. In each case the number of classes is three.

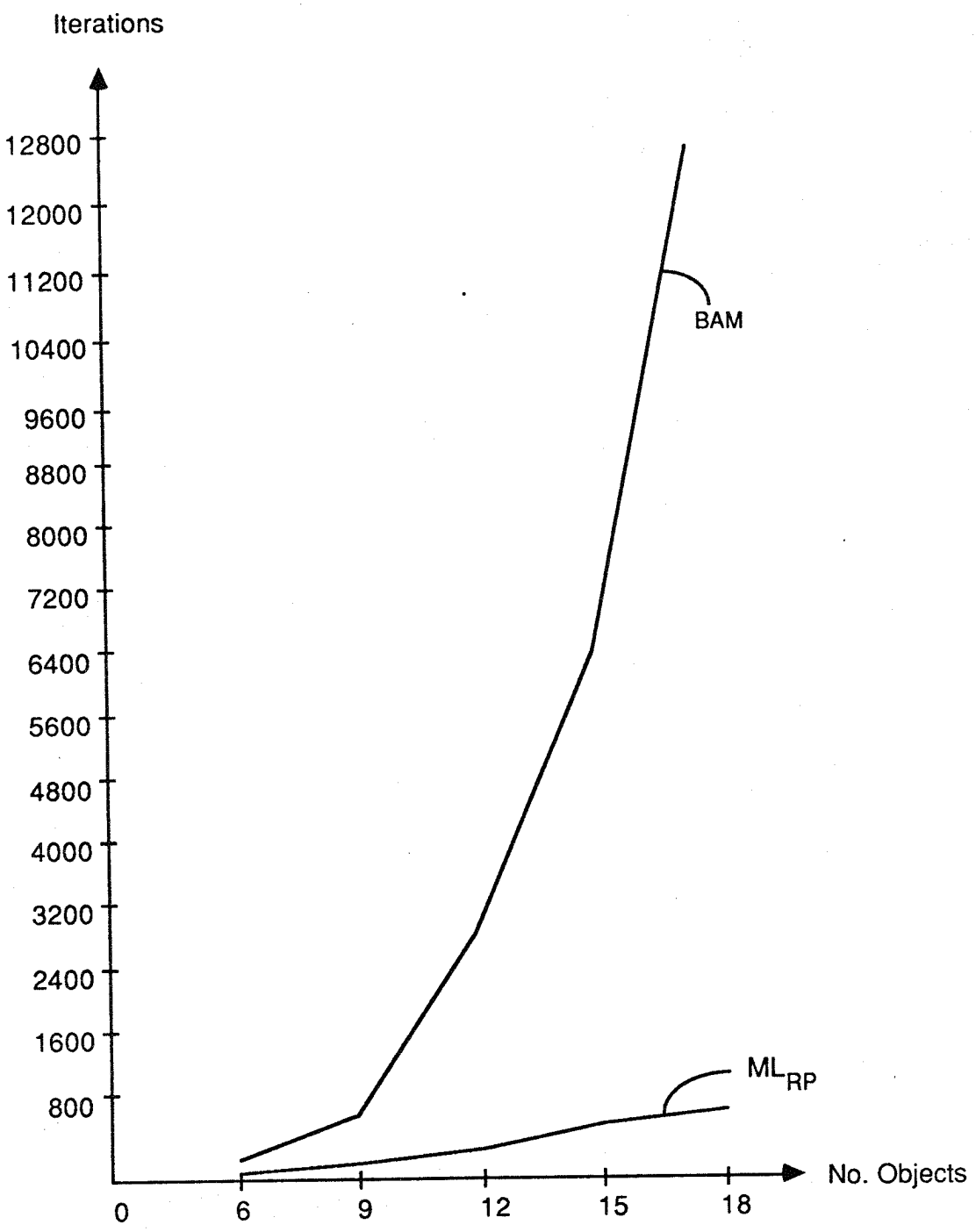


**Figure 4** Plot of the average number of iterations required for convergence for the BAM and the $ML_{RP}$ as the number of objects increases. In each case the number of classes is three.

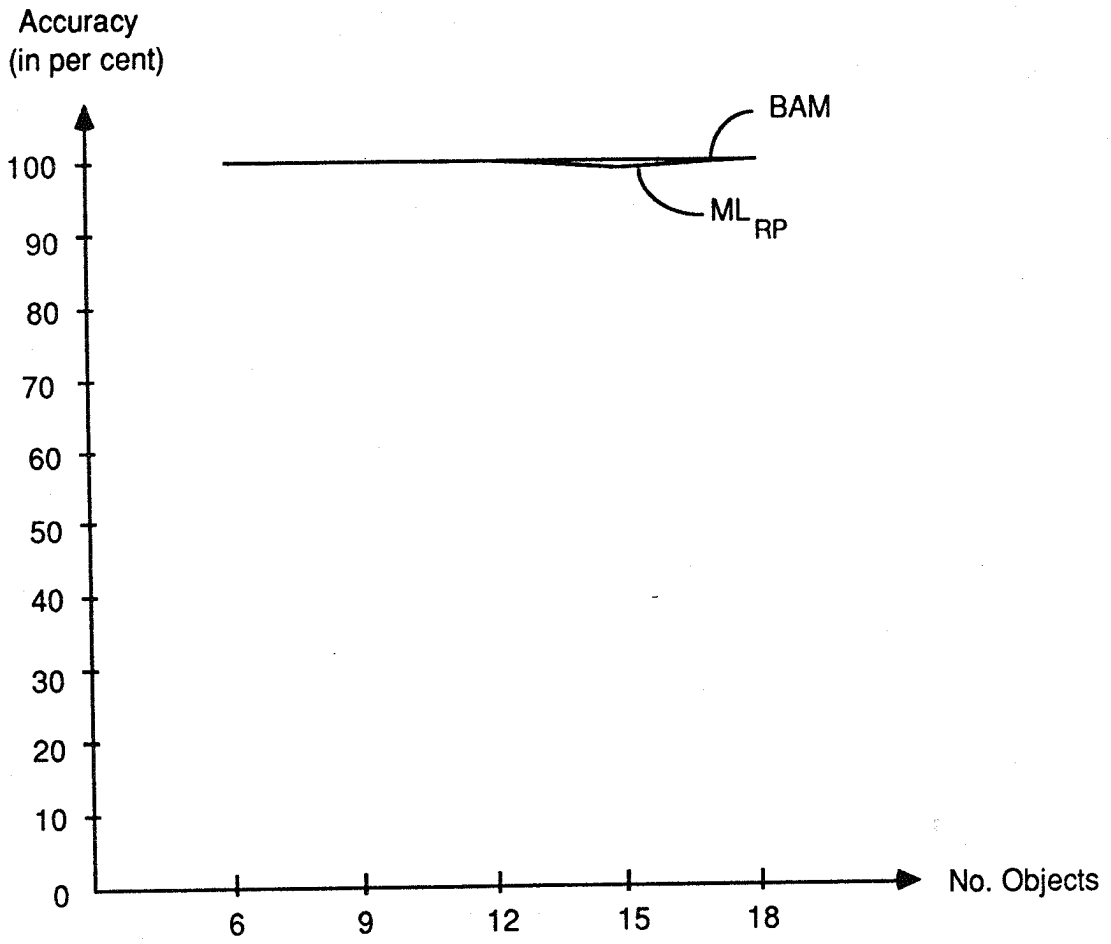


**Figure 5** Plot of the accuracy of the BAM and the $ML_{RP}$ automaton as a function of the the number of objects. In each case the number of classes is three.

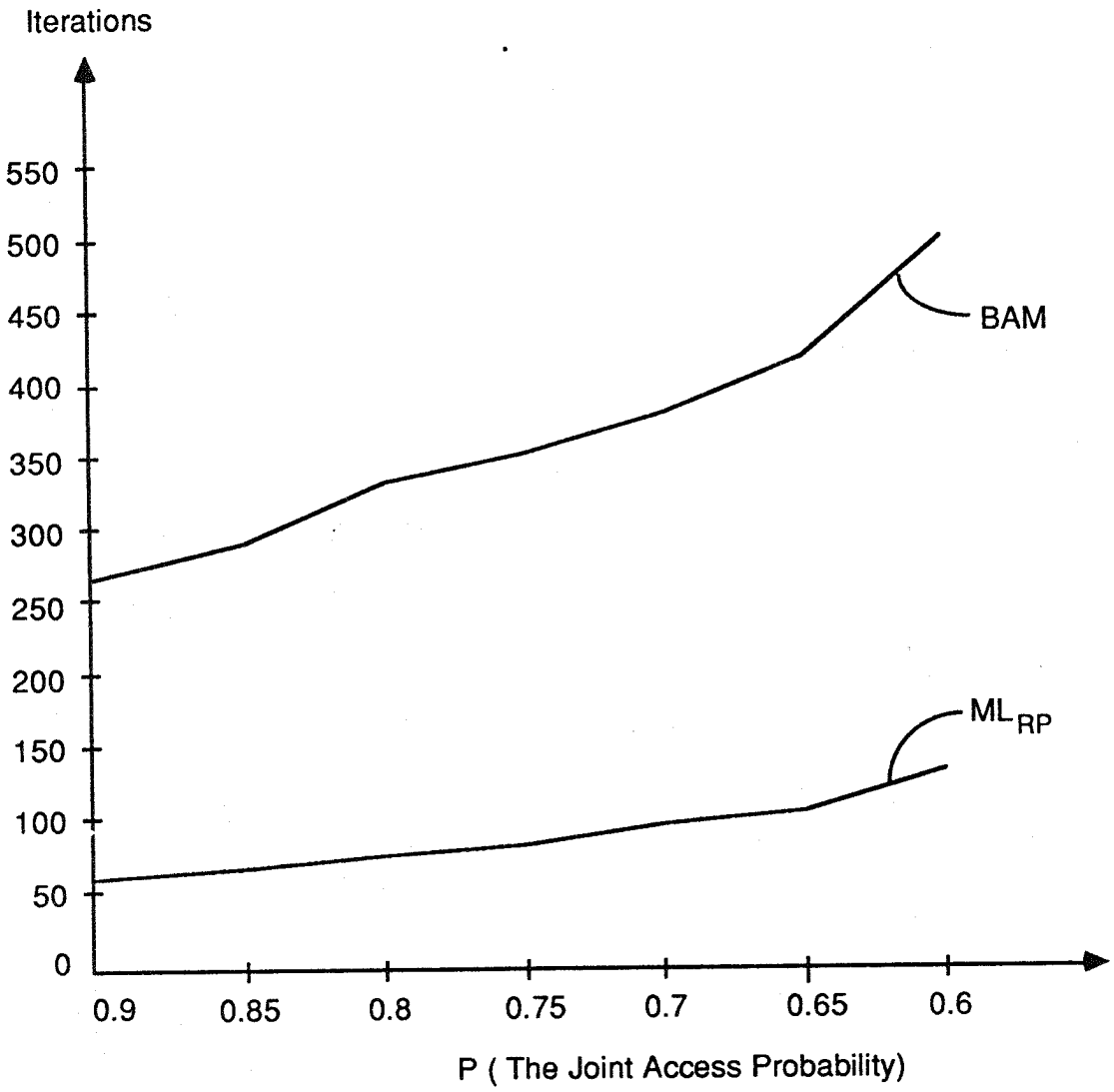


**Figure 6** Plot of the average number of iterations required for convergence for the BAM and the $ML_{RP}$ automaton as a function of the joint access probability P. In this case the number of objects is six and the number of classes is three.

# Carleton University, School of Computer Science
# Bibliography of Technical Reports
## Publications List

**School of Computer Science**
**Carleton University**
**Ottawa, Ontario, Canada**
**KIS 5B6**

SCS-TR-1 **The Design of CP-6 Pascal**
Jim des Rivieres and Wilf R. LaLonde, June 1982

SCS-TR-2 **Single Production Elimination in LR(1) PARSERS: A Synthesis**
Wilf R. LaLonde, June 1982

SCS-TR-3 **A Flexible Compiler Structure That Allows Dynamic Phase Ordering**
Wilf R. LaLonde and Jim des Rivieres, June 1982

SCS-TR-4 **A Practical Longest Common Subsequence Algorithm for Text Collation**
Jim des Rivieres, June 1982

SCS-TR-5 **A School Bus Routing and Scheduling Problem**
Wolfgang Lindenberg, Frantisek Fiala, July 1982

SCS-TR-6 **Routing Without Routing Tables: Labelling and Implicit Routing in Networks**
Nicola Santoro and Ramez Khatib, July 1982

SCS-TR-7 **Concurrency Control in Large Computer Networks**
Out-of-print Nicola Santoro and Jeffrey B. Sidney, July 1982

SCS-TR-8 **Order Statistics on Distributed Sets**
Out-of-print Nicola Santoro and Jeffrey B. Sidney, July 1982

SCS-TR-9 **Oligarchical Control of Distributed Processing Systems**
Moshe Krieger and Nicola Santoro, August 1982

SCS-TR-10 **Communication Bounds for Selection in Distributed Sets**
Nicola Santoro and Jeffrey B. Sidney, September 1982

SCS-TR-11 **A Simple Technique for Converting from a Pascal Shop to a C Shop**
Wilf R. LaLonde and John R. Pugh, November 1982

SCS-TR-12 **Efficient Abstract Implementations for Relational Data Structures**
Nicola Santoro, December 1982

SCS-TR-13 **On The Message Complexity of Distributed Problems**
Nicola Santoro, December 1982

SCS-TR-14 **A Common Basis for Similarity Measures Involving Two Strings**
R.L. Kashyap and B.J. Oommen, January 1983

# Carleton University, School of Computer Science
# Bibliography of Technical Reports

SCS-TR-84 **Deterministic Learning Automata Solutions to the Object Partitioning Problem**
B. John Oommen, D.C.Y. Ma, November 1985

SCS-TR-85 **Selecting Subsets of the Correct Density**
M.D. Atkinson, December 1985

SCS-TR-86 **Robot Navigation in Unknown Terrains Using Learned Visiblity Graphs. Part I: The Disjoint Convex Obstacles Case**
B. J. Oommen, S.S. Iyengar, S.V.N. Rao, R.L. Kashyap, February 1986

SCS-TR-87 **Breaking Symmetry in Synchronous Networks**
Greg N. Frederickson, Nicola Santoro, April 1986

SCS-TR-88 **Data Structures and Data Types: An Object-Oriented Approach**
John R. Pugh, Wilf R. LaLonde and David A. Thomas, April 1986

SCS-TR-89 **Ergodic Learning Automata Capable of Incorporating Apriori Information**
B. J. Oommen, May 1986

SCS-TR-90 **Iterative Decomposition of Digital Systems and Its Applications**
Vaclav Dvorak, May 1986.

SCS-TR-91 **Actors in a Smalltalk Multiprocessor: A Case for Limited Parallelism**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-92 **ACTRA - A Multitasking/Multiprocessing Smalltalk**
David A. Thomas, Wilf R. LaLonde, and John R. Pugh, May 1986

SCS-TR-93 **Why Exemplars are Better Than Classes**
Wilf R. LaLonde, May 1986

SCS-TR-94 **An Exemplar Based Smalltalk**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-95 **Recognition of Noisy Subsequences Using Constrained Edit Distances**
B. John Oommen, June 1986

SCS-TR-96 **Guessing Games and Distributed Computations in Synchronous Networks**
J. van Leeuwen, N. Santoro, J. Urrutia and S. Zaks, June 1986.

SCS-TR-97 **Bit vs. Time Tradeoffs for Distributed Elections in Synchronous Rings**
M. Overmars and N. Santoro, June 1986.

SCS-TR-98 **Reduction Techniques for Distributed Selection**
N. Santoro and E. Suen, June 1986.

SCS-TR-99 **A Note on Lower Bounds for Min-Max Heaps**
A. Hasham and J.-R. Sack, June 1986.

# Carleton University, School of Computer Science
# Bibliography of Technical Reports

SCS-TR-32 **Multi-Action Learning Automata Possessing Ergodicity of the Mean**
_____ B.J. Oommen and M.A.L. Thathachar, August 1983.

SCS-TR-33 **Fibonacci Graphs, Cyclic Permutations and Extremal Points**
Out-of-print N. Santoro and J. Urrutia, December 1983

SCS-TR-34 **Distributed Sorting**
_____ D. Rotem, N. Santoro, and J.B. Sydney, December 1983.

SCS-TR-35 **A Reduction Technique for Selection in Distributed Files: II**
_____ N. Sanatoro, M. Scheutzow, and J.B. Sidney, December 1983.

SCS-TR-36 **The Asymptotic Optimality of Discretized Linear Reward-Inaction Learning Automata**
_____ B.J. Oommen and Eldon Hansen, January 1984.

SCS-TR-37 **Geometric Containment is Not Reducible to Pareto Dominance**
_____ N. Santoro, J.B. Sidney, S.J. Sidney, and J. Urrutia, January 1984.

SCS-TR-38 **An Improved Algorithm for Boolean Matrix Multiplication**
_____ N. Santoro and J. Urrutia, January 1984.

SCS-TR-39 **Containment of Elementary Geometric Objects**
_____ J. Sack, N. Santoro and J. Urrutia, February 1984

SCS-TR-40 **SADE: A Programming Environment for Designing and Testing Systolic Algorithms**
_____ J.P. Corriveau and N. Santoro, February 1984.

SCS-TR-41 **Intersection Graphs, $\{B_1\}$-Orientable Graphs and Proper Circular Arc Graphs**
_____ Jorge Urrutia, February 1984.

SCS-TR-42 **Minimum Decompositions of Polygonal Objects**
_____ J. Mark Keil and Jörg-R. Sack, March 1984.

SCS-TR-43 **An Algorithm for Merging Heaps**
_____ Jörg-R. Sack and Thomas Strothotte, March 1984.

SCS-TR-44 **A Digital Hashing Scheme for Dynamic Multiattribute Files**
_____ E.J. Otoo, March 1984

SCS-TR-45 **Symmetric Index Maintenance Using Multidimensional Linear Hashing**
_____ E.J. Otoo, March 1984

SCS-TR-46 **A Mapping Function for the Directory of a Multidimensional Extendible Hashing**
_____ E.J. Otoo, March 1984.

SCS-TR-47 **Translating Polygons in the Plane**
_____ Jörg-R. Sack, March 1984.

SCS-TR-48 **Constrained String Editing**
_____ J. Oommen, May 1984.