An Application of Array Token Petri Nets to Clustering Analysis for Syntactic Patterns

D. Lalitha
Department of Mathematics
Sathyabama University
Chennai-119, India

K. Rangarajan
Department of MCA
Bharath University
Chennai-73, India

ABSTRACT
A new approach for clustering analysis using array token Petri net structure is introduced**.** We define two kinds of distances between an array R and a group of arrays characterized by Array Token Petri Net. A clustering procedure is proposed.

Keywords
Augmented Petri net, core Petri net, column shift, distance, firing sequence, row shift.

1. INTRODUCTION
The theoretical concepts of array grammars have been used effectively in the field of syntactic character recognition [1, 2]. Cluster analysis can be performed on a set of patterns on the basis of a selected similarity measure. The results of cluster analysis can then be used directly for Pattern Recognition. Similarity measure for syntactic patterns in terms of grammar transformations has been proposed. Distance between arrays in terms of the error transformations required to derive one array from another has already been defined in literature. Application of array grammars to clustering analysis exists in literature [3, 4]. A clustering procedure for English characters has been proposed by defining a distance between an array and a group of arrays characterized by an array grammar.

On the other hand Petri net [5] is an abstract formal model of information flow. Motivated by the fact that Array Token Petri Net Structure (ATPNS) has been defined to generate most of the two dimensional languages generated by array grammars [6, 7, 8, 9, 10], we have tried to use ATPNS in clustering. Here we propose an algorithm to generate distorted English characters. First we start with the Petri Net that generates the alphabet in perfect ratio. Add more transitions to the Petri Net to get the Core Petri Net which generates a noiseless, pure character of the alphabet which may not maintain a perfect ratio. Adding extra transitions to the Core Petri net enables it to generate some variations or noisy alphabets. The Petri Net generating noisy characters are called Augmented Petri Net [11]. Then we define two kinds of distances between an array $\mathcal{R}$ and a group of arrays characterized by Array Token Petri Net. We introduce a clustering procedure to be used in character recognition. In this paper we have used only one primitive **'x'**. **'.'** is used to denote a blank.

This paper is organized as follows. Section 2 gives all the basic definitions and notations used. Section 3 defines the distance measure between arrays and also gives the clustering procedure.

2. ARRAY TOKEN PETRI NET
In this section we define the basic concepts of Petri net, Array Token Petri Net Structure, Core Petri Net, Augmented Petri Net and give some examples.

Definition: 2.1

A Petri Net structure is a four tuple C = (P, T, I, O) where P = $\{p_1, p_2, \ldots, p_n\}$ is a finite set of places, $n \geq 0$, T = $\{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions $m \geq 0$, $P \cap T = \emptyset$, $I : T \rightarrow P^{\infty}$ is the input function from transitions to bags of places and $O : T \rightarrow P^{\infty}$ is the output function from transitions to bags of places.

Definition: 2.2

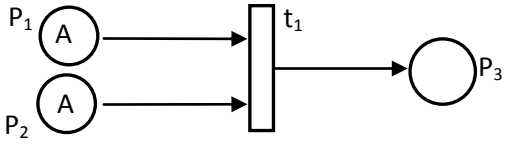
A Petri Net marking is an assignment of tokens to the places of a Petri Net. The tokens are used to define the execution of a Petri Net .The number and position of tokens may change during the execution of a Petri Net. In this paper arrays over an alphabet are used as tokens.

Basic Notations: Σ^{**}denotes the arrays made up of elements of Σ. If A and B are two arrays having same number of rows then A ϖ B is the column wise catenation of A and B. If two arrays have the same number of columns then AΘB is the row wise catenation of A and B. $(x)^n$ denotes a horizontal sequence of n 'x' and $(x)_n$ denotes a vertical sequence of n 'x' where $x \in \Sigma^{**}$. $(x)^{n+1} = (x)^n \varpi x$ and $(x)_{n+1} = (x)_n \Theta x$. We use the symbol $\oplus$ to denote either ϖ or Θ.

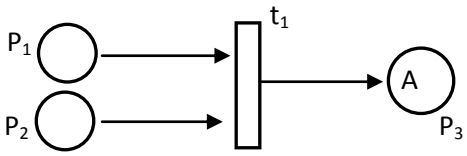
Catenation Rule as label for transitions: Column catenation rule is in the form A ϖ B. Here the array A denotes the m×n array in the input place of the transition. B is an array language whose number of rows will depend on 'm' the number of rows of A. The number of columns of B is fixed. For example A ϖ $\begin{pmatrix} x & x \end{pmatrix}_m$ adds two columns of x after the last column of the array A which is in the input place. But $\begin{pmatrix} x & x \end{pmatrix}_m$ ϖ A would add two columns of x before the first column of A. 'm' always denotes the number of rows of the input array A. Row catenation rule is in the form A Θ B. Here again the array A denotes the mxn array in the input place of the transition. B is an array language whose number of columns will depend on 'n' the number of columns of A. The number of rows of B is always fixed. For example A Θ $\begin{bmatrix} x \\ x \end{bmatrix}^n$ adds two rows of x after the last row of the array A which is in the input place. But $\begin{bmatrix} x \\ x \end{bmatrix}^n$ Θ A would add two rows of x before the first row of the array A. 'n' always denotes the number of columns of the input array A.

Enabled transitions: If all the input places of a transition have the same array as tokens, the transition becomes enabled. On firing it removes the array from the input places and moves it into all its output places.

Position of tokens before firing the transition

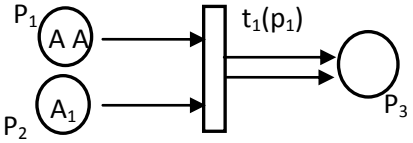


Position of tokens after firing the transition

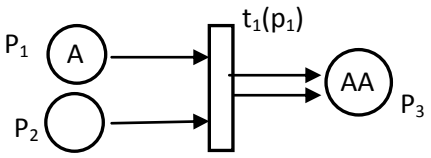


If all the input places of a transition have different arrays as token or if one input place has different arrays as tokens then the transition without a label is not enabled. The label has to specify one of its input places. The transition becomes enabled if it has an input place as label and that input place has at least the required number of tokens of the same array. Firing the transition removes an array from all its input places and puts the array, in the place specified by the label, into all its output places.

Position of tokens before firing the transition

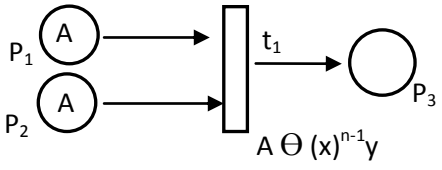


Firing the transition moves the array in place p_1(specified in the label of t_1) to p_3.



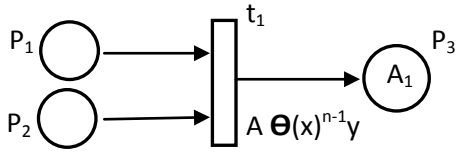
If all the input places of a transition have the same array as a token, the transition becomes enabled. If the transition has a catenation rule as label then firing it removes the array from the input places (denoted by A) catenates the array according to the rule and moves it into all its output places. If the transition has many input places and all the input places have different arrays as token then the transition with catenation rule as label cannot fire. If the transition has one input place with different arrays as token then the transition with catenation rule as label cannot fire.

Position of arrays before firing the enabled transition



$$\text{If A} = \begin{matrix} a & a & a \\ a & a & a \\ a & a & a \end{matrix}$$

Firing the transition puts A_1 in place p_3.



$$A_1 = \begin{matrix} a & a & a \\ a & a & a \\ a & a & a \\ x & x & y \end{matrix}$$ Firing t_1 adds a row $x \quad x \quad y$ as n =3

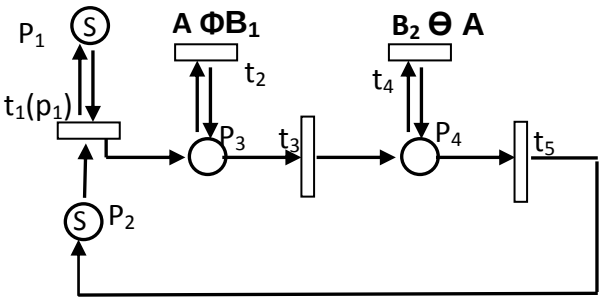
Definition 2.3:
If C = (P, T, I,O) is a Petri net structure with arrays over of Σ^{**} as initial markings, M0 : P $\rightarrow \Sigma^{**}$, label of at least one transition being catenation rule and a finite set of final places F $\subset$ P, then the Petri net structure C is defined as Array Token Petri Net Structure (ATPNS).

Definition 2.4:
If C is a ATPNS then the language generated by the Petri net C is defined as L(C) = {A ϵ Σ^{**} / A is in p for some p in F}.

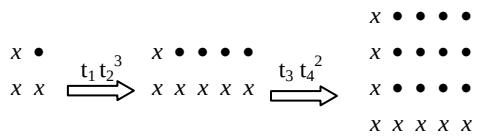
With arrays of Σ^{**} in some places as tokens or initial marking, all possible sequences of transitions are fired. Firing a transition either just moves the array or catenates according to the rule and then moves the array to the output place. Arrays reaching the final places are collected to form the language generated by C.

Definition 2.5:
If C is an ATPNS, let $\mathcal{B}$ be the collection of array languages used in the catenation rules of the labels attached to the transitions of C. We define the Core Petri Net (CPN) as the tuple (C, $\mathcal{B}$). Let us see an example of a CPN generating pure L. By varying the firing sequences we get the class of arrays in different sizes and different ratios.

Example 2.1:
Core Petri Net (C, $\mathcal{B}$) to generate a class of noiseless **L**. The Petri Net C is given in the figure



$S = \begin{matrix} x & \bullet \\ x & x \end{matrix}$, $B_1 = \begin{matrix} (\bullet)_{m-1} \\ x \end{matrix}$, $B_2 = x\ (\bullet)^{n-1}$, $\mathcal{B}$ = { B_1, B_2}, F = { p_6 }.
The firing sequence $t_1t_2^3t_3t_4^2t_5$ derives the following array



3. DISTANCE MEASURE BETWEEN ARRAYS

In this section we define row shift, column shift and also two kinds of distances between an array $\mathcal{R}$ and a group of arrays characterized by the Augmented Array Token Petri Net.

Definition 3.1:
Let A and B are two arrays over a {x}. Let $1 \le i \le m$, $1 \le j \le n$. If $a_{ij} = x$ with $b_{ij} \neq x$ and if there exists k with $1 \le k \le n$ such that $a_{ik} \neq x$ with $b_{ik} = x$ then define the row shift $R_{ij}(A,B) = \min |j-k|$, where the minimum is taken over for all such k. If no such k exists then find k , with $1 \le k \le m$ such that $a_{kj} \neq x$ and $b_{kj} = x$, then define the column shift $C_{ij}(A,B) = \min |j-k|$, where the minimum is taken over for all such k. If $a_{ij} = b_{ij} = x$ then both $R_{ij}(A,B)$ and $C_{ij}(A,B)$ are zero.

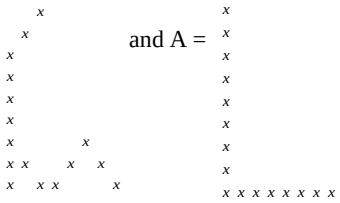
If 'x' is missing in the ij th position look for it in the ith row (an extra 'x'), if not look for it in jth column (an extra x). If no such k is found either in the same row or column then both the row shift R_{ij} and column shift C_{ij} are undefined. No 'x' in any position should be taken for both row shift as well as column shift.

Definition 3.2:
Let A and B are two arrays over a {x}. The distance measure between A and B is defined as

$$D(A,B) = \sum_{i=1}^{m}\sum_{j=1}^{n} R_{ij}(A,B) + C_{ij}(A,B)$$

if, for $1 \le i \le m$, $1 \le j \le n$, all $R_{ij}(A,B)$ and $C_{ij}(A,B)$ are properly defined. Otherwise we define $D(A,B) = \infty$.
Let us explain with an example.

If $\mathcal{R}$ =



and A =

then $R_{11} = 2$, $R_{21} = 1$, $C_{92} = 1$, $C_{95} = 1$, $C_{96} = 2$ and $C_{97} = 1$. All other row shifts and column shifts are zero. $D(A, \mathcal{R}) = 3 + 5 = 8$. There is no undefined row or column shift in this example.

We assume that every array $\mathcal{R}$ is a variation of a pure noiseless array $\mathcal{A}$, generated by some noiseless, pure array generated by a core Petri net (C, $\mathcal{B}$). The array token Petri net structure uses the concept of column and row catenation to generate English alphabets. Without loss of generality we can assume that we add a single row or single column at a time when a transition is fired. The various array languages used as label in generation of English alphabets are as follows.

Arrays used in row catenation rules $x(\bullet)^{n-1}$, $(\bullet)^{n-1}x$, $(\bullet)^n$, $(x)^n$, $x(\bullet)^{n-2}x$. If $\mathcal{B}$ has the array language $x(\bullet)^{n-1}$ then add $\bullet x(\bullet)^{n-2}$ to the augmented array language list $\mathcal{B}^a$. If $\mathcal{B}$ has the array $(\bullet)^{n-1}x$ language then add $(\bullet)^{n-2}x\bullet$ to the augmented array language list $\mathcal{B}^a$. If $\mathcal{B}$ has the array language $x(\bullet)^{n-2}x$ then add $\bullet x(\bullet)^{n-3}x$ and $x(\bullet)^{n-3}x\bullet$ to the augmented array language list $\mathcal{B}^a$.
Arrays used in column catenation rules

$$\begin{matrix} x \\ (\bullet)_{m-1} \end{matrix}, \quad \begin{matrix} (\bullet)_{m-1} \\ x \end{matrix}, \quad \begin{matrix} x \\ (\bullet)_{m-2} \\ x \end{matrix}, \ (\bullet)_m, \ (x)_m .$$

If $\mathcal{B}$ has the array language $\begin{matrix} x \\ (\bullet)_{m-1} \end{matrix}$ then add $\begin{matrix} \bullet \\ x \\ (\bullet)_{m-2} \end{matrix}$ to the augmented array language list $\mathcal{B}^a$. If $\mathcal{B}$ has the array language $\begin{matrix} (\bullet)_{m-1} \\ x \end{matrix}$ then add $\begin{matrix} (\bullet)_{m-2} \\ x \\ \bullet \end{matrix}$ to the augmented array language list $\mathcal{B}^a$. If $\mathcal{B}$ has the array language $\begin{matrix} \bullet \\ x \\ (\bullet)_{m-2} \\ x \end{matrix}$ then add $\begin{matrix} x \\ (\bullet)_{m-3} \\ x \end{matrix}$ and $\begin{matrix} x \\ (\bullet)_{m-3} \\ x \\ \bullet \end{matrix}$ to the augmented array language list $\mathcal{B}^a$.

Add the extra array languages to $\mathcal{B}$ to form $\mathcal{B}^\alpha$. Add extra transitions, with catenation rules using the augmented array languages as labels, to the CPN to form the augmented Petri net structure C^a. Then Augmented Petri Net (APN) is the tuple (C^a, $\mathcal{B}^\alpha$). The given $\mathcal{R}$ may belong to the language generated by APN. We now give the algorithm to construct APN given a CPN.

Algorithm:

Input. Core Petri Net (C, $\mathcal{B}$) where C = (P, T, I, O) with P = $\{p_1, p_2, ...p_q\}$ is a finite set of places, $q \ge 0$, T = $\{t_1, t_2, \ldots, t_l\}$ is a finite set of transitions $l \ge 0$, $P \cap T = \emptyset$, $I: T \to P^\infty$ is the input function from transitions to bags of places and $O: T \to P^\infty$ is the output function from transitions to bags of places. $\mathcal{B}$ ={ B_1, $B_2, \ldots, B_r$}the collection of array languages used in the labels of the transitions of C.

Output. Augmented Petri Net (C^a, $\mathcal{B}^\alpha$) where C^a = (P^a,T^a,I^a,O^a) with P^a is a finite set of places, $T^a = \{t_1, t_2, \ldots, t_l, t_1', t_2', \ldots t_k'\}$ is a finite set of transitions, $P \cap T = \emptyset$, I^a: $T \to P^\infty$ is the input function from transitions to bags of places and $O^a: T \to P^\infty$ is the output function from transitions to bags of places. $\mathcal{B}^a$ ={ $B_1, B_2, \ldots, B_r, B_1', B_2', .. , B_k'$} the collection of array languages used in the labels of the transitions of C^a.

Start:

Step 1. Let $P^a = P$, $T^a = T$, $I^a = I$, $O^a = O$, $\mathcal{B}^a = \mathcal{B}$.

Step 2. For i = 1, .., r repeat steps 3 to step 11.

Step 3. If B_i is a row matrix with 'x' only at b_{11} then define B_i' as a row matrix with 'x' only at b_{12}. Add B_i' to $\mathcal{B}^a$. Go to step 9.

Step 4. If B_i is a row matrix with 'x' only at b_{1n} then define B_i' as a row matrix with 'x' only at b_{1n-1}. Add B_i' to $\mathcal{B}^a$. Go to step 9.

Step 5. If B_i is a row matrix with 'x' at b_{11} and b_{1n} then define B_i' as a row matrix with 'x' at b_{12} and b_{1n}. Define B_i'' as a row matrix with 'x' at b_{11} and b_{1n-1}. Add both B_i' and B_i'' to $\mathcal{B}^a$. Go to step 9.

Step 6. If B_i is a column matrix with 'x' only at b_{11} then define B_i' as a column matrix with 'x' only at b_{21}. Add B_i' to $\mathcal{B}^a$. Go to step 9.

Step 7. If B_i is a column matrix with 'x' only at b_{n1} then define $B_i^{'}$ as a column matrix with 'x' only at b_{n-11}. Add $B_i^{'}$ to $\mathcal{B}^a$. Go to step 9.

Step 8. If B_i is a column matrix with 'x' at b_{11} and b_{n1} then define $B_i^{'}$ as a column matrix with 'x' at b_{21} and b_{n1}. Define $B_i^{''}$ as a column matrix with 'x' at b_{11} and b_{n-11}. Add both $B_i^{'}$ and $B_i^{''}$ to $\mathcal{B}^a$. Go to step 9.

Step 9. For j = 1,2,…*l* repeat step 10.

Step 10. If t_j is the transition which has a catenation rule $A \oplus B_i$ involving B_i, with input place p and output place q then add a transition $t_j^{'}$ ($t_j^{''}$) to T^a. Label of $t_j^{'}$ is the catenation rule $A \oplus B_i^{'}$ ($A \oplus B_i^{''}$), with input place p and output place q. Add the transitions $t_j^{'}$ and $t_j^{''}$ (if it exists) in T^a.

Step 11. For every $t_j^{'}$ ($t_j^{''}$), $1 \leq j \leq \ell$, added in step 9 $I^a(t_j^{'}) = I^a(t_j)$ and $O^a(t_j^{'}) = O^a(t_j)$ ($I^a(t_j^{''}) = I^a(t_j)$ and $O^a(t_j^{''}) = O^a(t_j)$). Add these extra input and output functions arising due to the addition of extra transitions to I^a and O^a. Increase I value by 1 and go to step 3.

Step 12. $C^a = (P^a, T^a, I^a, O^a)$ and $\mathcal{B}^a = \{B_1, B_2, \ldots, B_r, B_i^{'}, B_i^{''}, \ldots\}$.
End.

We use the algorithm to construct the APN for the CPN given in example 2.1.

Example 3.1:
Core Petri Net (C, $\mathcal{B}$) to generate the character **L** is given in example 2.1

Step 1. Let $P^a = \{p_1, p_2, p_3, p_4\}$, $T^a = \{t_1, t_2, t_3, t_4, t_5\}$, $I^a(t_1) = \{p_1, p_2\}$, $I^a(t_2) = \{p_3\}$ $I^a(t_3) = \{p_3\}$, $I^a(t_4) = \{p_4\}$, $I^a(t_5) = \{p_4\}$, $O^a(t_1) = \{p_1, p_3\}$, $O^a(t_2) = \{p_3\}$ $o^a(t_3) = \{p_4\}$, $O^a(t_4) = \{p_4\}$, $O^a(t_5) = \{p_2\}$, $\mathcal{B}^a = \{B_1, B_2\}$.

Step 2. Since r = 2, i takes values 1 and 2.
Since B_1 is a column matrix with b_{n1} = x we go to step 7.

Step 7. Define $B_1^{'} = \begin{matrix} \bullet \\ x \\ (\bullet)_{m-2} \end{matrix}$. Add $B_1^{'}$ to $\mathcal{B}^a$.

Step 9. j takes values from 1 to 5.

Step 10. We find the transition t_2 with the label $A \Phi B_1$, $I(t_2) = \{p_3\}$ and $O(t_2) = \{p_3\}$ involving the array language B_1. Add in the net the transition $t_2^{'}$ with label $A \varphi B_1^{'}$. Add $t_2^{'}$ to T^a.

Step 11. Add $I^a(t_2^{'}) = \{p_3\}$, $O^a(t_2^{'}) = \{p_3\}$ to the input output functions respectively. Increase I to 2. and go to step 3.
Since B_2 is a row matrix with b_{1n} = x we go to step 4.

Step 4. $B_2 = (\bullet)^{n-1} x$ hence define $B_2^{'} = (\bullet)^{n-2} x \bullet$. Add $B_2^{'}$ to $\mathcal{B}^a$.

Step 9. j takes values 1 to 5.

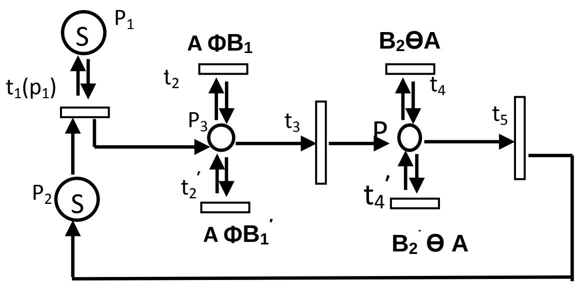
Step 10. t_4 is the transition with label $B_2 \Theta A$, $I(t_4) = \{p_4\}$ and $O(t_4) = \{p_4\}$. Add in the net the transition $t_4^{'}$ with label $B_2^{'} \Theta A$. Add $t_4^{'}$ to T^a.

Step 11. Add $I^a(t_4^{'}) = \{p_4\}$, $O^a(t_4^{'}) = \{p_4\}$ to the input output functions respectively.

Step 12. $T^a = T^a \cup \{t_2^{'}, t_4^{'}\}$, $I^a = I^a \cup \{I^a(t_2^{'}) = \{p_3\}, I^a(t_4^{'}) = \{p_4\}\}$, $O^a = O^a \cup \{O^a(t_2^{'}) = \{p_3\}, O^a(t_4^{'}) = \{p_4\}\}$. $C^a = (P^a, T^a, I^a, O^a)$, $\mathcal{B}^a = \mathcal{B}^a \cup \{B_1^{'}, B_2^{'}\}$.

Now let us draw the graph of the Augmented Petri Net.

$S = \begin{matrix} x & \bullet \\ x & x \end{matrix}$, $B_1 = \begin{matrix} (\bullet)_{m-1} \\ x \end{matrix}$, $B_2 = x\,(\bullet)^{n-1}$, $B_1^{'} = \begin{matrix} (\bullet)_{m-2} \\ x \\ \bullet \end{matrix}$ and $B_2^{'} = \bullet x (\bullet)^{n-2}$ to $\mathcal{B}^a$. The graph of APN is given below.



$S = \begin{matrix} x & \bullet \\ x & x \end{matrix}$, $B_1 = \begin{matrix} (\bullet)_{m-1} \\ x \end{matrix}$, $B_1^{'} = \begin{matrix} \bullet \\ x \\ (\bullet)_{m-2} \end{matrix}$, $B_2 = x\,(\bullet)^{n-1}$,

$B_2^{'} = \bullet x (\bullet)^{n-2}$

This Augmented Petri Net which generates both pure **L** and noisy **L**. If the firing sequence does not contain any augmented transitions then the **L** generated will be noiseless. The firing sequence $t_1\, t_2^2\, t_2^{'}\, t_2\, t_3 t_4^3\, t_4^{'}\, t_4^2\, t_5$, which has some augmented transitions, will generate the following distorted **L**.

$$\begin{matrix} x & \bullet & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & x & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & \bullet & \bullet \\ x & \bullet & \bullet & \bullet & x & \bullet \\ x & x & x & x & \bullet & x \end{matrix}$$

Now we are in a position to define the distance between an array and a group of arrays characterized by both CPN and APN.

Definition 3.3:
Distance between an array $\mathcal{R}$ and a group of arrays characterized by a APN C^a, denoted by $d_a(C^a, \mathcal{R})$ = min D(A, $\mathcal{R}$) where the minimum is taken over all arrays A generated by C^a.

Definition 3.4:
Distance between an array $\mathcal{R}$ and a group of arrays characterized by a CPN C, denoted by $d_c(C, \mathcal{R}) = d_a(C^a, \mathcal{R})$ + minimum number of augmented transitions required to generate the array A whose distance with $\mathcal{R}$ is minimum.

Let us explain the definition with a given $\mathcal{R}$ and the arrays generated by the firing sequences of the net given in example 3.1.

$\mathcal{R}$ =

```
      x
    x
  x
  x
  x
  x
  x          x
  x x      x   x
  x   x x        x
```

The firing sequence $t_1 t_2^2\, t_2^{'3}\, t_2 t_3 t_4^5\, t_4^{'2}\, t_5$ generates the array

A =

```
   x
   x
 x
 x
 x
 x
 x
 x       x x x
 x x x x       x
```

and D(A, $\mathcal{R}$) = 3 which is the minimum and so $d_a(C^a, \mathcal{R})$ =3.

Given an array $\mathcal{R}$ and C_k, $1 \leq k \leq n$ a set of CPN generating various English alphabets, construct the corresponding set of

APN. For $1 \leq k \leq n$, evaluate d_a (C_k^a, $\mathcal{R}$) for all APN constructed. Assign the array $\mathcal{R}$ in to the cluster generated by the C_k^a for which d_a (C_k^a, $\mathcal{R}$) is minimum, provided k is unique. Note that for more than one value of k the distance d_a (C_k^a, $\mathcal{R}$) could be minimum. If that is the case for those values of k evaluate d_c(C_k, $\mathcal{R}$). Classify the array R in to the cluster generated by the C_k for which d_c (C_k, $\mathcal{R}$) is minimum.

4. CONCLUSION

In this paper we have proposed an algorithm to derive an Augmented Petri Net from the Core Petri Net to generate irregular English Alphabets. We have defined a distance between a given array and a group of arrays generated by the Augmented Petri Net and Core Petri Net. With this distance definition we have given a clustering procedure to be used in character recognition.

5. REFERENCES

[1] H. Fernau, R. Freund and M. Holzer, Character recognition with *k*-head finite array automata**,** Advances in Pattern Recognition, Lecture Notes in Computer Science, 1998, Volume 1451, pp 282-291.

[2] H. Fernau and R. Freund, Bounded Parallelism in Array Grammars used for Character Recognition, Advances in Structural and Syntactical Pattern Recognition, Lecture Notes in Computer Science, 1996, Volume 1121, pp 40 - 49.

[3] Patrick Shen-Pei Wang, An Application Of Array Grammars To Clustering Analysis For Syntactic Patterns, Pattern Recognition Vol. 17, No. 4, 1984, pp. 441- 451.

[4] K.S. Fu, S.Y.Lu, A Clustering Procedure for Syntactic Patterns, IEEE Trans. Syst. Man and Cybernet. SMC-7, 1977, 734- 742

[5] James L. Peterson, Petri Net Theory and Modeling of systems, Prentice Hall, Englewood Cliffs, N J, 1981.

[6] D. Lalitha and K. Rangarajan, Adjunct array token Petri nets, Proceedings of International Conference on Operations Research Applications in Engineering and Management (ICOREM), Anna University, Tiruchirapalli, 2009, pp. 431–445.

[7] D. Lalitha and K. Rangarajan, Column and Row Catenation Petri Net Systems, Proceeding of Fifth IEEE International Conference on Bio-Inspired Computing: Theories and Applications, 2010, pp.1382–1387.

[8] D. Lalitha and K. Rangarajan, Characterization of Pasting System Using Array Token Petri Nets, International Journal of Pure and Applied Mathematics, Vol 70, issue 3, 2009, pp 24-29.

[9] D. Lalitha, K. Rangarajan, D.G. Thomas, "Petri Net Generating Hexagonal Arrays," LNCS 6636, 2011, pp. 235–247, Proceedings of IWCIA 2011, The Fourteenth International Workshop on Combinatorial Image Analysis.

[10] D. Lalitha, K. Rangarajan, D.G. Thomas, Petri Net Generating Context-Sensitive Hexagonal Array Languages, International Conference on Mathematical and Computational Models, 2011, pp 389-397.

[11] D. Lalitha and K. Rangarajan, Augmented Petri Nets Generating Irregular English Alphabets, Proceedings of the National Conference on Mathematical Techniques and its Applications, 2012, pp 204-212.