# A Data Partitioning Approach to speed up the Fuzzy ARTMAP algorithm using the Hilbert space–filling Curve

José Castro
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816–2786
e-mail: jcastro@pegasus.cc.ucf.edu

Michael Georgiopoulos
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816–2786
e-mail: michaelg@mail.ucf.edu

Ronald Demara
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816–2786
e-mail: demara@pegasus.cc.ucf.edu

Avelino Gonzalez
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816–2786
e-mail: gonzalez@pegasus.cc.ucf.edu

***Abstract*— One of the properties of FAM, which can be both an asset and a liability, is its capacity to produce new neurons (templates) on demand to represent classification categories. This property allows FAM to automatically adapt to the database without having to arbitrarily specify network structure. This same property though has the undesirable side effect that, on large databases, it can produce a large network size that can dramatically slow down the algorithms training time. To address this problem we propose the use of the Hilbert space–filling curve. Our results indicate that the Hilbert space–filling curve can reduce the training time of FAM by partitioning the training set into smaller sub-sets and have many FAMs trained, each one of them on a different training sub-set. This approach does not affect the classification performance or the network size. On the positive side, it speeds up the convergence time, required by FAM, to learn large datasets. Given that there is full data partitioning with the HSFC we implement and test a parallel implementation on a Beowulf cluster of workstations that further speeds up the training and classification time on large databases.**

## I. Introduction

Neural Networks have been used extensively and successfully to address a wide variety of problems. As computing capacity and electronic databases grow, there is an increasing need to process considerably larger datasets. In this context, the algorithms of choice tend to be ad–hoc algorithms or tree based algorithms such as CART and C4.5 [9]. Variations of these tree learning algorithms, such as SPRINT (Shafer, et al., [10]) and SLIQ (Mehta, et al., [7]) have been successfully adapted to handle very large data sets.

Neural network algorithms have, for some applications, prohibitively high convergence times. Even one of the fastest neural network algorithms, the Fuzzy ARTMAP (FAM) algorithm, tends to lag in convergence time as the size of the network grows. The FAM algorithm corresponds to a family of neural network architectures introduced by Carpenter, et al., 1991-1992 [3], [2] and has proven to be one of the premier neural network architectures for classification problems. Some of the advantages that FAM has, compared to other neural network classifiers, are that it learns the required task fast, it has the capability to do on-line learning, and its learning structure allows the explanation of the answers that the neural network produces.

One of FAM's properties which is a mixed blessing, is its capacity to produce new neurons (*templates*) on demand to represent classification categories. This property allows FAM to automatically adapt to the database without having to arbitrarily and a-priori specify its network structure, but it also has the undesirable side effect that on large databases it can produce a large network size that can dramatically slow down the algorithm's training time. It would be desirable to have a method capable of keeping FAM's convergence time manageable, without affecting the generalization performance of the network or its resulting size when the training is completed.

In this paper we propose the use of the Hilbert space–filling curves for partitioning the training set to be used with FAM classification (hFAM). Our research on Hilbert space–filling curves has shown that they can dramatically reduce the training time of FAM by partitioning the training set without a significant effect on the classification (generalization) performance of the network or the network size. Skopal et al. [4] analyze different space–filling curves, amongst them the Peano curve, Z curve and the Hilbert curve, and also provide measures for their appropriateness. Moon et al. [8]

argue and prove that the Hilbert space–filling curve is the mapping that provides the *least number of splits* of compact sets from $[0,1]^n$ to [0,1]. This can be interpreted as stating that points that are close on the mapping will also be close on the n–dimensional hypercube. Lawder [6] has taken advantage of this result and used the Hilbert space–filling curves to develop a multi–dimensional indexing technique.

This paper is organized as follows: First we review the Fuzzy ARTMAP architecture and functionality, then we examine the computational complexity of FAM and analyze how and why a partitioning approach can considerably reduce the algorithm's training time. After that we discuss space–filling curves in general and the Hilbert space–filling curve in particular and why this curve can be instrumental in improving the FAM algorithm's convergence time. Furthermore, experimental results are presented on a sequential machine and on a Beowulf cluster of workstations that illustrate the merit of our approach. We close the paper with a summary of the findings and suggestions for further research.

## II. The Fuzzy ARTMAP Architecture

The Fuzzy ARTMAP architecture consists of four layers or fields of nodes (see Figure 1). The layers that are worth describing are the *input layer* ($F_1^a$), the *category representation layer* ($F_2^a$), and the *output layer* ($F_2^b$). The input layer of Fuzzy ARTMAP is the layer where an input vector of dimensionality $2M_a$ of the following form is applied

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) = (a_1, a_2, \ldots, a_{M_a}, a_1^c, a_2^c, \ldots, a_{M_a}^c) \qquad (1)$$

where:

$$a_i^c = 1 - a_i;\ \forall i \in \{1, 2, \ldots, M_a\} \qquad (2)$$

The assumption here is that the input vector **a** is such that each one of its components lies in the interval [0, 1]. The layer $F_2^a$ of Fuzzy ARTMAP is referred to as the *category representation layer*, because this is where categories (or groups) of input patterns are formed. Finally, the output layer is the layer that produces the outputs of the network. An output of the network represents the output to which the input applied at the input layer of FAM is supposed to be mapped to.

There are two sets of weights worth mentioning in FAM. The first set of weights are weights from $F_2^a$ to $F_1^a$, designated as $w_{ji}^a, (1 \le j \le N_a, 1 \le i \le 2M_a)$, and referred to as top-down weights. The vector of weights $\mathbf{w}_j^a = \left(w_{j1}^a, w_{j2}^a, \ldots, w_{j,2M_a}^a\right)$ is called a *template*. Its functionality is to represent the group of input patterns that chose node $j$ in the category representation layer of Fuzzy ARTMAP as their representative node. The second set of weights, worth mentioning, are weights that emanate from every node $j$ in the category representation layer to every node $k$ in the output layer. These weights are designated as $W_{jk}^{ab}$ (called inter–ART weights). The vector of inter–ART weights emanating from every node $j$ in Fuzzy ARTMAP $\left(i.e., \mathbf{W}_j^{ab} = \left[W_{j1}^{ab}, W_{j2}^{ab}, \ldots, W_{j,N_b}^{ab}\right]\right)$ corresponds to the output pattern that this node j is mapped to.

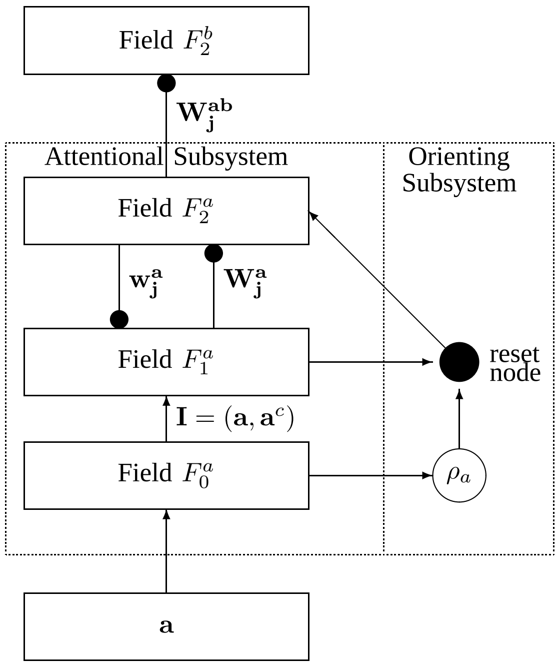


Fig. 1. Fuzzy ARTMAP Diagram

Fuzzy ARTMAP can operate in two distinct phases: the *training phase* and the *performance phase*. The training phase of Fuzzy ARTMAP can be described as follows: Given a list of input/output pairs, $\left\{(\mathbf{I}^1, \mathbf{O}^1), \ldots, (\mathbf{I}^r, \mathbf{O}^r), \ldots, (\mathbf{I}^{PT}, \mathbf{O}^{PT})\right\}$, we want to train Fuzzy ARTMAP to map every input pattern of the training list to its corresponding output pattern. To achieve the aforementioned goal we present the training list to Fuzzy ARTMAP architecture repeatedly. That is, we present $\mathbf{I}^1$ to $F_1^a$, $\mathbf{O}^1$ to $F_2^b$, $\mathbf{I}^2$ to $F_1^a$, $\mathbf{O}^2$ to $F_2^b$, and finally $\mathbf{I}^{PT}$ to $F_1^a$, and $\mathbf{O}^{PT}$ to $F_2^b$. We present the training list to Fuzzy ARTMAP as many times as it is necessary for Fuzzy ARTMAP to correctly classify all these input patterns. The task is considered accomplished (i.e., the learning is complete) when the weights do not change during a list presentation. The aforementioned training scenario is called *off–line learning*. The performance phase of Fuzzy ARTMAP works as follows: Given a list of input patterns, such as $\tilde{\mathbf{I}}^1, \tilde{\mathbf{I}}^2, \ldots, \tilde{\mathbf{I}}^{PS}$, we want to find the Fuzzy ARTMAP output produced when each one of the aforementioned test patterns is presented at its $F_1^a$ layer. In order to achieve the aforementioned goal we present the test list to the trained Fuzzy ARTMAP architecture and we observe the network's output.

The operation of Fuzzy ARTMAP is affected by two network parameters, the choice parameter $\beta_a$, and the baseline vigilance parameter $\bar{\rho}_a$. The choice parameter takes values in the interval $(0, \infty)$, while the baseline vigilance parameter assumes values in the interval [0,1]. Both of these parameters affect the number of nodes created in the category representation layer of Fuzzy ARTMAP. Higher values of $\beta_a$ and $\bar{\rho}_a$ create more nodes in the category representation layer of

Fuzzy ARTMAP, and consequently produce less compression of the input patterns. There are two other network parameter values in Fuzzy ARTMAP that are worth mentioning. The vigilance parameter $\rho_a$, and the number of nodes $N_a$ in the category representation layer of Fuzzy ARTMAP. The vigilance parameter $\rho_a$ takes value in the interval $[\bar{\rho}_a, 1]$ and its initial value is set to be equal to $\bar{\rho}_a$ . The number of nodes $N_a$ in the category representation layer of Fuzzy ARTMAP increases while training the network and corresponds to the number of committed nodes in Fuzzy ARTMAP plus one uncommitted node.

Before training the top–down weights (the $w^a_{ji}$'s) of Fuzzy ARTMAP are set equal to 1, and the inter–ART weights (the $W^{ab}_{jk}$'s) are chosen equal to 0. One of the specific operands involved in all of these operations is the *fuzzy min operand*, designated by the symbol $\wedge$. Actually, the fuzzy min operation of two vectors $x$, and $y$, designated as $x \wedge y$, is a vector whose components are equal to the minimum of components of $x$ and $y$. Another specific operand involved in these equations is designated by the symbol $|\cdot|$. In particular, $|x|$ is the size of a vector $x$ and is defined to be the sum of its components.

## A. The Fuzzy ARTMAP Learning Algorithm

The FAM learning algorithm consists of three major operations:

**Operation 1:** Calculation of bottom up inputs to every node $j$ in $F^a_2$, as follows:

$$T^a_j = \frac{|\mathbf{I}^r \wedge \mathbf{w}^a_j|}{\beta_a + |\mathbf{w}^a_j|} \tag{3}$$

after calculation of the bottom up inputs the node $j_{max}$ with the maximum bottom up input is chosen.

**Operation 2:** The node $j_{max}$ with the maximum bottom up input is examined to determine whether it passes the vigilance criterion. A node passes the vigilance criterion if the following condition is met:

$$\frac{|\mathbf{I}^r \wedge \mathbf{w}^a_j|}{|\mathbf{I}^r|} \geq \rho_a \tag{4}$$

if the vigilance criterion is satisfied we proceed with operation 3 otherwise node $j_{max}$ is disqualified and we find the next node in sequence in $F^a_2$ that maximizes the bottom up input. Eventually we will end up with a node $j_{max}$ that maximizes the bottom up input and passes the vigilance criterion.

**Operation 3:** This operation is implemented only after we have found a node $j_{max}$ that maximizes the bottom-up input of the remaining nodes in competition and that passes the vigilance criterion. Operation 3 determines whether this node $j_{max}$ passes the prediction test. The prediction test checks if the inter–ART weight vector emanating from node $j_{max}$ $\left(i.e., \mathbf{W}^{ab}_{j_{max}} = \left[W^{ab}_{j_{max}1}, W^{ab}_{j_{max}2}, \dots, W^{ab}_{j_{max},N_b}\right]\right)$ matches exactly the desired output vector $\mathbf{O}^r$ (if it does this is referred to as *passing the prediction test*). If the node does not pass the prediction test, the vigilance parameter $\rho_a$ is increased to the level of $\frac{|\mathbf{I}^r \wedge \mathbf{w}^a_j|}{|\mathbf{I}^r|} + \varepsilon$ where $\varepsilon$ is a very small number, node $j_{max}$ is disqualified, and the next in sequence node that maximizes the bottom-up input and passes the vigilance is chosen. If, on the other hand, node $j_{max}$ passes the predictability test, the weights in Fuzzy ARTMAP are modified as follows:

$$\mathbf{W}^a_{j_{max}} \leftarrow \mathbf{W}^a_{j_{max}} \wedge \mathbf{I}^r, \quad \mathbf{W}^{ab}_{j_{max}} \leftarrow \mathbf{O}^r \tag{5}$$

Fuzzy ARTMAP training is considered complete if and only if after repeated presentations of all training input/output pairs to the network (where Operations 1-3 are implemented)no weight changes are produced. In some databases noise in the data may create over-fitting so a single pass over the training set may be preferable. In the performance phase of Fuzzy ARTMAP only Operations 1 and 2 are implemented for every input pattern presented to it.

## B. Fuzzy ARTMAP pseudocode

We will be interested in classification problems where we associate input patterns to category labels. The FAM pseudo-code pertaining to the case where FAM is used for classification problems, is provided below.

```
FAM-Training-Phase($Patterns, \bar{\rho}, \beta_a, \varepsilon$)
 1  $templates \leftarrow \{\}$
 2  for each $I$ in $Patterns$
 3     do $\rho \leftarrow \bar{\rho}$
 4        repeat
 5              $T_{max} \leftarrow \frac{M_a}{2M_a+\beta_a}$
 6              $status \leftarrow$ FoundNone
 7              for each $w_j$ in $templates$
 8                 do if $\rho(I, w^a_j) \geq \rho_a$
 9                       and $T(I, w^a_j, \beta_a) > T_{max}$
10                      then
11                           $T_{max} \leftarrow T(I, w^a_j, \beta_a)$
12                           $j_{max} \leftarrow j$
13                           $status \leftarrow$ FoundOne
14              if $status =$ FoundOne
15                 then if $class(I) = class(w_{j_{max}})$
16                         then $status \leftarrow$ ThisIsIT
17                         else  $status \leftarrow$ TryAgain
18                               $\rho \leftarrow \rho(I, w^a_{j_{max}}) + \varepsilon$
19        until $status \neq$ TryAgain
20        if $status =$ ThisIsIT
21           then
22              $w^a_{j_{max}} \leftarrow w^a_{j_{max}} \wedge I$
23           else
24              $templates \leftarrow templates \cup \{I\}$
25
26  return $templates$
```

where $T(I, w^a, \beta)$ is defined by equation 3 and $\rho(I, w^a)$ is defined by the left hand side of inequality 4.

## C. FAM time complexity analysis

If we call $\Gamma$ the average number of times that the **repeat** loop (see pseudo-code) is executed for each input pattern. Then the number of times that a given input pattern $I$ passes through the code will be:

$$Time(I) = O(\Gamma \times |Templates|) \tag{6}$$

Also, under the artificial condition that the number of templates does not change during training it is easy to see that the time complexity of the algorithm is:

$$Time(FAM) = O(\Gamma \times PT \times |Templates|) \tag{7}$$

Usually for a fixed type of database the FAM algorithm achieves a certain *compression ratio*. This means that the number of templates is actually a fraction of the number of patterns $PT$ in the training set:

$$|Templates| = \kappa PT \tag{8}$$

and

$$Time(FAM) = O\left(\Gamma PT \kappa PT\right) = O(\kappa \Gamma PT^2) \tag{9}$$

Dividing the training set into $p$ partitions will reduce the number of patterns in each partition to $\frac{PT}{p}$ and the number of templates in each partition to $\frac{\kappa PT}{p}$, on the average. On a sequential machine the speedup obtained by partitioning the training set into $p$ subsets will be proportional to:

$$\frac{T_1}{T_p(sequential)} = \frac{\kappa \Gamma PT^2}{p\Gamma \frac{\kappa PT}{p}\frac{PT}{p}} = p \tag{10}$$

and on a parallel machine with $p$ processors the speedup will be proportional to:

$$\frac{T_1}{T_p(parallel)} = \frac{\kappa \Gamma PT^2}{\Gamma \frac{\kappa PT}{p}\frac{PT}{p}} = p^2 \tag{11}$$

## III. Space–Filling Curves

A space filling curve $\mathcal{S}_m^{M_a}$ can be thought of as an $m^{th}$–order approximation of the space–filling curve $\mathcal{S}$ in the $M_a$–dimensional space. An $M_a$–dimensional space–filling curve with grid size N connects $N^{M_a}$ points and has $N^{M_a} - 1$ segments. Figure 2 shows the Sweep and Peano space–filling curves respectively. The grid size in these examples is 4, the number of dimensions $M_a = 2$ the number of points that they connect is $4^2 = 16$, and the number of segments is 15.

A curve $\mathcal{S}$ is space–filling iff:

$$\mathcal{S} \overset{\text{def}}{=} \lim_{m\to\infty} \mathcal{S}_m^{M_a} = [0,1]^{M_a} \tag{12}$$

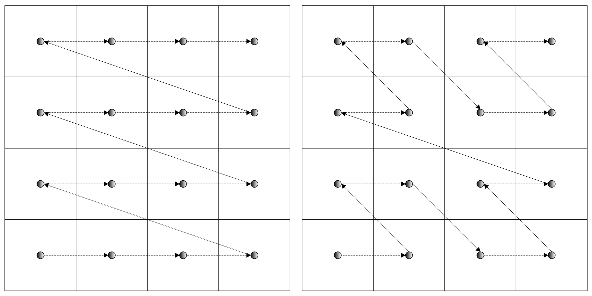


Fig. 2. Sweep and Peano space–fi lling curves

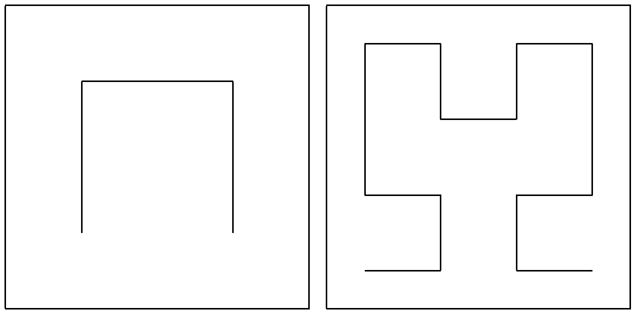


Fig. 3. First 2 approximations of HSFC

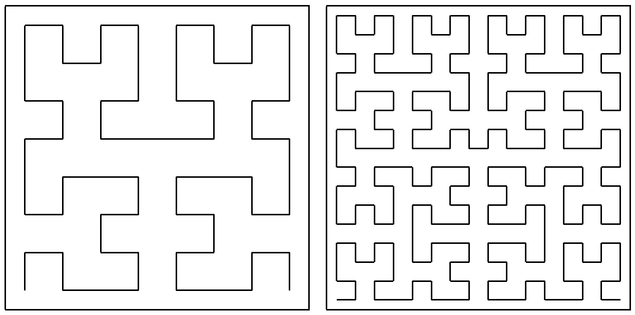


Fig. 4. Hilbert space–fi lling Curve

### A. The Hilbert space–filling Curve

We will denote the $m^{th}$–order approximation of the $M_a$–dimensional Hilbert space–filling curve as $\mathcal{H}_m^{M_a}$. Examples of the first 4 approximations of the 2–dimensional Hilbert space–filling curve can be seen in figures 3 and 4. The $m^{th}$–order approximation $\mathcal{H}_m^{M_a}$ of the HSFC has a grid size of $N = 2^m$. In practice $\mathcal{H}_m^{M_a}$ divides and the $M_a$–dimensional space into $2^{mM_a}$ boxes and orders them in a contiguous sequence. For a more detailed exposition of the clustering properties of this curve we refer the reader to [8].

### B. The Hilbert space–filling curve for FAM partitioning

Our approach is the following: we take the set of training pairs $(\mathbf{I}^r, \mathbf{O}^r)$, apply the Hilbert index $r = \mathcal{H}_m^{M_a}(\mathbf{a})$, where $\mathbf{a}$ is the non complement coded part of $\mathbf{I} = (\mathbf{a}, 1 - \mathbf{a})$. The resulting values are added to an index file and sorted. Once sorted the index is split into $p$ contiguous and equal sized partitions, each partition is processed independently. The complexity of the partitioning operation is equal to the complexity of the sorting algorithm used. For any reasonable sort this is $O(PT \log(PT))$ and therefore does not add to the complexity of the FAM learning process itself (which is at least $O(PT^2)$).

## IV. Experimental Design

Experiments where conducted on 3 databases: 1 real dataset and 2 artificial Gaussian data sets. All data sets were tested with training set sizes of $1000 \times 2^i, i \in \{0, 1, \ldots, 9\}$, that is 1,000 to 512,000 patterns. The test set size was fixed at 20,000 patterns. The number of partitions varied from $p = 1$ (*vanilla*

FAM) to $p = 32$. Partition sizes increased in powers of 2. The tests where conducted 32 independent times for each different $(p, PT) =$ (partition, training set size) pair.

### A. Forest CoverType Database

The first database used for testing was the Forest CoverType database provided by Blackard [1], and donated to the UCI Machine Learning Repository. The number of attributes of each pattern is 54. There are no missing values on this data.

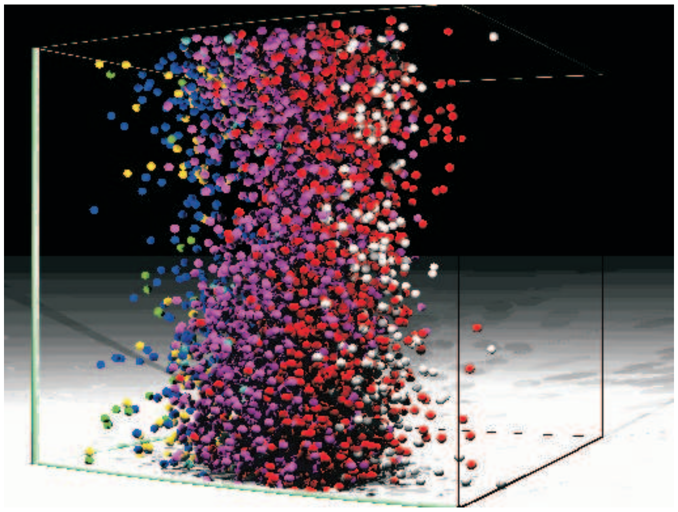


Fig. 5. Visualization of Forest CoverType data

Patterns 1 through 512,000 were used for training. The test set for all trials consisted of patterns 561,001 to 581,000. A visualization of the first 3 dimensions of the Forest Covertype can be seen in figure 5, different tones correspond to different classes. Classification performance for different machine learning algorithms for this database usually hovers around 75%.

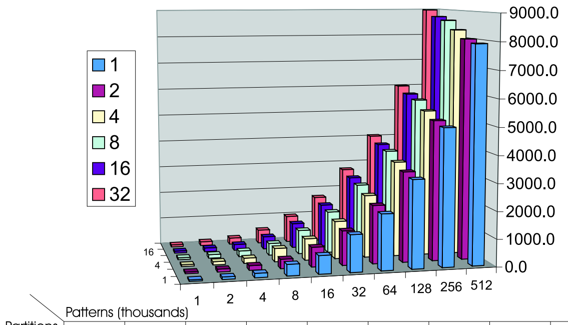
### B. Gaussian Databases

The Gaussian data was artificially generated using the polar form of the Box–Muller transform with the R250 random number generator by Kirkpatrick and Scholl [5]. We generated 2 class 16-dimensional data. 532,000 patterns where generated for each Gaussian database. 512,000 patterns were used for training. The last 20,000 where used for testing. One Gaussian database had a 15% overlap, while the other had a 5% overlap.

## V. Experimental Results

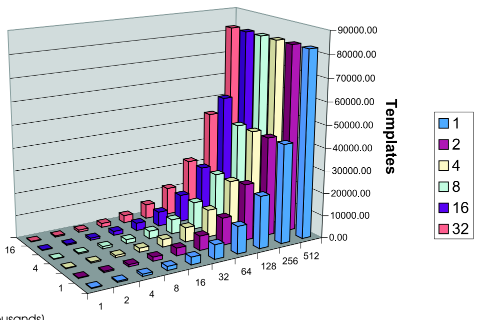
In figure 6 we can see a bar graph of the number of templates on the Z axis, the training set size on the X axis (in thousands of patterns), and the number of partitions on the Y axis. The data in figure 6 correspond to the Covertype database. From the graph it is evident that the number of partitions do not affect considerably the compression ratio. This is also confirmed in figures 7 for the Gaussian databases.

The classification (generalization) performance of FAM, using the HSFC partitioning approach, is very comparable to the classification performance of the non-partitioned FAM (as shown in figure 8). In figure 8. The Tree Covertype database classification continues to improve up to 512,000 patterns, clearly indicating that the database is sufficiently complex as to need a large training set size. This behavior is not observed on the Gaussian artificial data (5% or 15% data), where the performance peaks at 32,000 patterns.



| Partitions \ Patterns (thousands) | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| 🟦 1 | 50.0 | 90.4 | 156.8 | 410.3 | 663.9 | 1347.9 | 2031.8 | 3208.6 | 5004.7 | 7923.5 |
| 🟪 2 | 60.0 | 103.3 | 170.5 | 341.1 | 715.9 | 1257.3 | 2129.4 | 3303.8 | 5124.7 | 8037.5 |
| 🟨 4 | 63.0 | 117.3 | 188.4 | 489.4 | 790.4 | 1384.4 | 2298.4 | 3498.7 | 5364.3 | 8322.0 |
| 🟩 8 | 70.0 | 129.7 | 211.5 | 438.0 | 877.9 | 1528.3 | 2505.8 | 3755.0 | 5660.8 | 8637.6 |
| 🟦 16 | 74.0 | 138.2 | 220.7 | 463.5 | 920.2 | 1592.3 | 2609.3 | 3866.0 | 5789.8 | 8750.6 |
| 🟥 32 | 80.0 | 144.6 | 237.3 | 497.4 | 989.5 | 1698.7 | 2761.5 | 4055.4 | 6013.1 | 8986.0 |

Fig. 6. Number of Templates on Covertype Data



| Partitions \ Patterns (thousands) | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| 🟦 1 | 283.00 | 520.00 | 950.19 | 1781.22 | 3366.19 | 6401.75 | 12157.97 | 23099.66 | 43727.59 | 82609.69 |
| 🟪 2 | 283.00 | 520.00 | 950.19 | 1781.22 | 3366.19 | 6401.66 | 12158.06 | 24899.47 | 43729.19 | 82506.75 |
| 🟨 4 | 283.00 | 520.03 | 949.41 | 1781.88 | 3366.59 | 6401.53 | 12156.84 | 23101.50 | 43724.84 | 82498.59 |
| 🟩 8 | 275.19 | 520.03 | 949.53 | 1782.00 | 3367.03 | 6402.00 | 12156.16 | 23101.34 | 43722.59 | 82498.53 |
| 🟦 16 | 283.00 | 520.00 | 949.47 | 1782.34 | 3367.09 | 6401.69 | 12157.72 | 23100.72 | 53664.41 | 82506.00 |
| 🟥 32 | 283.00 | 520.00 | 949.69 | 1781.88 | 3366.88 | 6401.28 | 12156.28 | 23099.66 | 43723.78 | 82505.31 |

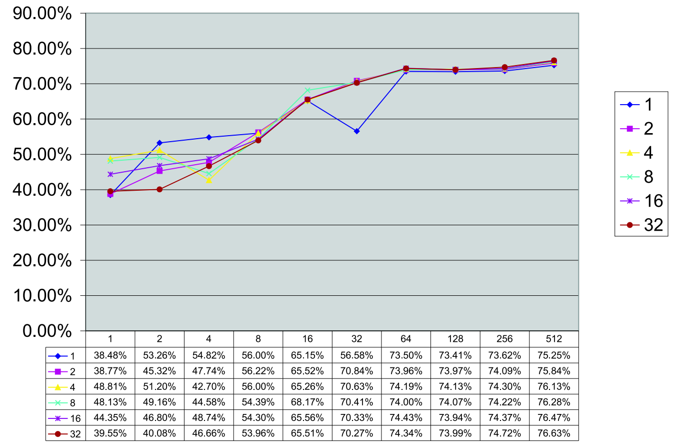
Fig. 7. Number of Templates on 5% Overlap Gaussian Data

Figure 9 shows the speedup of the partitioned FAM with $p$ partitions running in parallel using equation 10. The best speedups obtained where in the order of 100.

The speedup for the same data using a sequential processing machine can be seen in figure 10. In this figure, we can see that the speedup for an single processor is in the order of $p$ (17 for 32 processors), as indicated by equation 10, and the speedup of the parallel implementation is in the order of $p^2$ (100 for 32 processors), as indicated by equation 11.

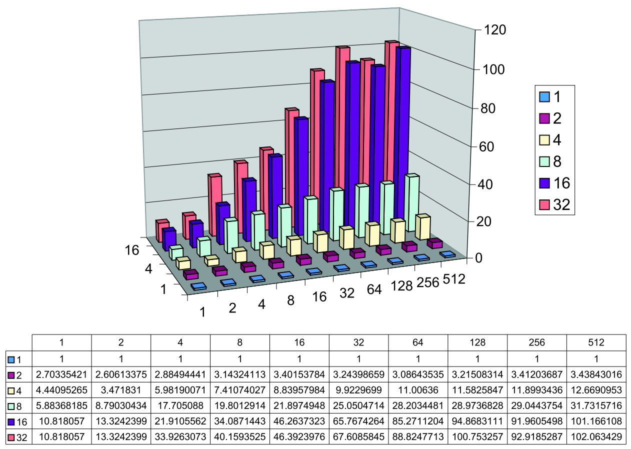
## VI. Conclusions

We considered the FAM algorithm as applied to classification tasks on large databases. We observed that its training time tends to slow considerably when the size of the database grows. After analyzing the algorithm we proposed the use



| | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 38.48% | 53.26% | 54.82% | 56.00% | 65.15% | 56.58% | 73.50% | 73.41% | 73.62% | 75.25% |
| 2 | 38.77% | 45.32% | 47.74% | 56.22% | 65.52% | 70.84% | 73.96% | 73.97% | 74.09% | 75.84% |
| 4 | 48.81% | 51.20% | 42.70% | 56.00% | 65.26% | 70.63% | 74.19% | 74.13% | 74.30% | 76.13% |
| 8 | 48.13% | 49.16% | 44.58% | 54.39% | 68.17% | 70.41% | 74.00% | 74.07% | 74.22% | 76.28% |
| 16 | 44.35% | 46.80% | 48.74% | 54.30% | 65.56% | 70.33% | 74.43% | 73.94% | 74.37% | 76.47% |
| 32 | 39.55% | 40.08% | 46.66% | 53.96% | 65.51% | 70.27% | 74.34% | 73.99% | 74.72% | 76.63% |

Fig. 8. Classifi cation Performance of Forest Cover database



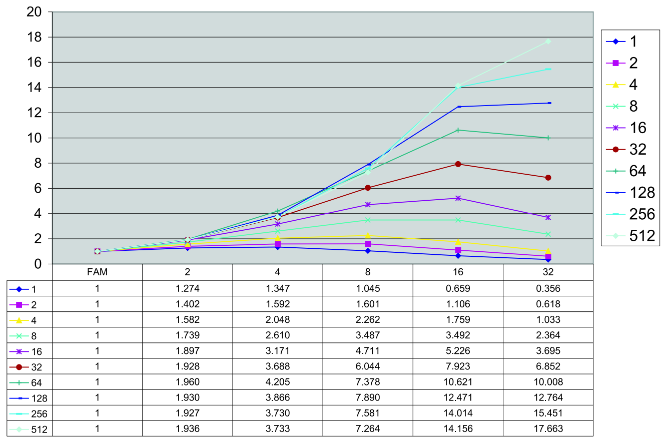
| | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2.70335421 | 2.60613375 | 2.88494441 | 3.14324113 | 3.40153784 | 3.24398659 | 3.08643535 | 3.21508314 | 3.41203687 | 3.43843016 |
| 4 | 4.44095265 | 3.471831 | 5.98190071 | 7.41074027 | 8.83957984 | 9.9229699 | 11.00636 | 11.5825847 | 11.8993436 | 12.6690953 |
| 8 | 5.88368185 | 8.79030434 | 17.705088 | 19.8012914 | 21.8974948 | 25.0504714 | 28.2034481 | 28.9736828 | 29.0443754 | 31.7315716 |
| 16 | 10.818057 | 13.3242399 | 21.9105562 | 34.0871443 | 46.2637323 | 65.7674264 | 85.2711204 | 94.8683111 | 91.9605498 | 101.166108 |
| 32 | 10.818057 | 13.3242399 | 33.9263073 | 40.1593525 | 46.3923976 | 67.6085845 | 88.8247713 | 100.753257 | 92.9185287 | 102.063429 |

Fig. 9. Hilbert Parallel Partitioning Speedup on Covertype Data

of Hilbert space–filling curves (HSFC) to address its slow convergence speed. HSFC achieve this objective by partitioning the dataset into smaller datasets and by training different, and independently, FAMs on each one of the smaller datasets. Experimental results on 3 databases confirm our claims: Classification performance is not affected. Compression ratio is only slightly affected (becomes smaller). Most importantly, convergence time is improved linearly on the sequential machine and quadratically on the parallel machine. Nevertheless there is still room for improvement. For instance, by combining HSFC data-partitioning with a network partitioning approach would accomplish optimal workload balance in the parallel implementation; this is one of the directions of our immediate future research.

## Acknowledgment

The authors would like to thank the Institute of Simulation and Training and the Link Foundation Fellowship program for partially funding this project. This work was also supported in part by the National Science Foundation under grant no. 0203446.



| | FAM | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| 1 | 1 | 1.274 | 1.347 | 1.045 | 0.659 | 0.356 |
| 2 | 1 | 1.402 | 1.592 | 1.601 | 1.106 | 0.618 |
| 4 | 1 | 1.582 | 2.048 | 2.262 | 1.759 | 1.033 |
| 8 | 1 | 1.739 | 2.610 | 3.487 | 3.492 | 2.364 |
| 16 | 1 | 1.897 | 3.171 | 4.711 | 5.226 | 3.695 |
| 32 | 1 | 1.928 | 3.688 | 6.044 | 7.923 | 6.852 |
| 64 | 1 | 1.960 | 4.205 | 7.378 | 10.621 | 10.008 |
| 128 | 1 | 1.930 | 3.866 | 7.890 | 12.471 | 12.764 |
| 256 | 1 | 1.927 | 3.730 | 7.581 | 14.014 | 15.451 |
| 512 | 1 | 1.936 | 3.733 | 7.264 | 14.156 | 17.663 |

Fig. 10. Hilbert Sequential Partitioning Speedup on Covertype Data

## References

[1] J. A. Blackard, "Comparison of neural networks and discriminant analysis in predicting forest cover types," Ph.D. dissertation, Department of Forest Sciences, Colorado State University, 1999.

[2] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental learning of analog multidimensional maps," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 698–713, September 1992.

[3] G. A. Carpenter, S. Grossberg, and J. H. Reynolds, "Fuzzy ART: An adaptive resonance algorithm for rapid, stable classifi cation of analog patterns," in *International Joint Conference on Neural Networks, IJCNN'91*, vol. II, IEEE/INNS Inc. Seattle, Washington: IEEE–INNS–ENNS, 1991, pp. 411–416.

[4] *Properties of Space Filling Curves and Usage With UB– Trees*. High Fatra, Slovakia: ITAT 2002, 2002.

[5] S. Kirkpatrick and E. Stoll, "A very fast shift–register sequence random number generator," *Journal of Computational Physics*, vol. 40, pp. 517–526, 1981.

[6] J. K. Lawder and P. J. H. King, "Using space–filling curves for multi–dimensional indexing," *Lecture Notes in Computer Science*, vol. 1832, 2000.

[7] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A fast scalable classifier for data mining," in *Extending Database Technology*. Avignon, France: Springer, 1996, pp. 18–32. [Online]. Available: citeseer.nj.nec.com/mehta96sliq.html

[8] B. Moon, H. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space–filling curve," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, January 2001.

[9] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, California: Morgan Kaufmann, 1993.

[10] J. C. Shafer, R. Agrawal, and M. Mehta, "SPRINT: A scalable parallel classifier for data mining," in *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds. Bombay, India: Morgan Kaufmann, September 1996, pp. 544–555. [Online]. Available: citeseer.nj.nec.com/shafer96sprint.html