Criticality Dispersion in Swarms to Optimize N-tuples

M.A. Hannan Bin Azhar
Department of Electronics
University of Kent
Canterbury, Kent, UK
maha2@kent.ac.uk

Farzin Deravi
Department of Electronics
University of Kent
Canterbury, Kent, UK
f.deravi@kent.ac.uk

Keith Dimond
Department of Electronics
University of Kent
Canterbury, Kent, UK
krdkent@waitrose.com

ABSTRACT
Among numerous pattern recognition methods the neural network approach has been the subject of much research due to its ability to learn from a given collection of representative examples. This paper concerns with the optimization of a weightless neural network, which decomposes a given pattern into several sets of *n* points, termed n-tuples. A population-based stochastic optimization technique, known as Particle Swarm Optimization (PSO), has been used to select an optimal set of connectivity patterns to improve the recognition performance of such "n-tuple" classifiers. The original PSO was refined by combining it with a bio-inspired technique called the Self-Organized Criticality (SOC) to add diversity in the population for finding better solutions. The hybrid algorithms were adapted for the n-tuple system and the performance was measured in selecting better connectivity patterns. The aim was to improve the discriminating power of the classifier in recognizing handwritten characters by exploiting the criticality dispersion in the swarm population. This paper presents the implementation of the hybrid model in greater detail with the effect of criticality dispersion in finding better solutions.

Categories and Subject Descriptors
I.5.1 [**Pattern Recognition**]: Models- Neural nets.

General Terms
Algorithms, Performance and Design.

Keywords
Swarm intelligence, Self-Organized criticality, Weightless neural network, Machine Learning, Optimization, Pattern Recognition and classification.

1. INTRODUCTION
Pattern recognition as a field is extremely diversified and has been applied in many areas such as science, engineering, business, medicine etc. The aim of pattern recognition is to classify objects into identifiable categories or classes after extracting features from the data. This data may be numerical, pictorial, textural, linguistic or any combination of these categories. Numerous techniques for pattern recognition can be investigated in four general approaches of pattern recognition, as suggested in [13]: template matching, statistical techniques, structural techniques and neural networks (NNs).

The neural classification emulates the computational paradigm of the behaviour of neurones and their interconnections in human brain. Instead of recognizing a pattern by following a set of human-designed rules, as in the structural approaches, neural nets learn the underlying rules from a given collection of representative examples. Among neural network models, the weightless or n-tuple form of network [6] stands out due to its own advantages over a variety of pattern recognition algorithms [23]. Considerable research activities have focused on the n-tuple method, both regarding theoretical issues [15][23] as well as applications to real-world tasks [24]. Several applications of n-tuple-based networks to handwritten character recognition tasks have been reported.

Considerable research shows that by optimizing the connections of an n-tuple network the classification performance can be improved significantly [2][5][12][14]. Particle swarm optimization is a population-based stochastic optimization technique developed by Eberhart and Kennedy [17] in 1995, motivated from the simulation of social behaviour of bird flocking or fish schooling. Being successfully applied in many areas like function optimization, artificial neural network training [25] or fuzzy system control [9], the PSO seems to be a good candidate to find an optimal set of input maps for the n-tuple network [3]. The particle swarm searches optima in the solution space and shrinks the search area step by step. It refines its search by attracting the particles to positions with good solutions.

In order to be less susceptible to premature convergence, the maintenance of "diversity" in particle swarm is important [16][19]. One way to add diversity in PSO is to use the Self-Organized Criticality (SOC) [4]. Self-organized criticality has been found in a variety of phenomena such as earthquakes, volcanic activity, the game of life, landscape formation and stock markets. SOC describes how small amounts of external influence can occasionally lead to the big changes observed in complex systems. Self-Organized Criticality has been successfully applied to improve the performance of Evolutionary Algorithms. This was done with mass extinction and mutation operator control by Krink and Thomsen [18], where extinction zones were formed (3×3 rectangles). Mutated copies of currently best individual then substituted individuals in these extinction zones. SOC was also used in relation to spatial mating control [21], where most mates were immediate neighbours, but occasionally mates were selected from remote places. Occasional outbreeding improved the performance by counter balancing the effect of rigid

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

neighbourhood inbreeding. Other aspects of SOC have been described and applied to search problems by Boettcher and Paczuski [7]. Extending the PSO with SOC was found to be very promising in achieving faster convergence and reaching better solutions and the resulting algorithm was named as SOC-PSO [19]. This paper presents the experimental results regarding the effect of criticality dispersion in the swarm population to optimize an n-tuple classifier with the application to the handwritten character recognition task of the NIST database [28]. The remainder of the paper has been organized as follows:

Section 2 will introduce the n-tuple network. Application of particle swarm on n-tuple systems will be introduced in Section 3. The main idea of SOC will be described in Section 4. Implementation of the hybrid SOC-PSO algorithm for the n-tuples will be described in Section 5. Modelling of the criticality dispersion and the importance of criticality limit will also be explained here. Section 6 will present the experimental results. Finally Section 7 will conclude the paper.

2. N-TUPLE NETWORK

Although the n-tuple classifier is not famously popular compared to some other methods, such as multilayer perceptrons [20], the n-tuple classifier does have its own advantages over a variety of pattern recognition algorithms [23]. The networks based on the n-tuple method have two great strengths: they can be trained quickly and they can be implemented in conventional computers simply when compared to other equation solving and minimizing methods. The training of the basic classifier is a one-shot memorization process. These advantages come at the cost of recognition robustness. It has been shown that the n-tuple method can result in quite reasonable recognition performance if used with care [23]. The n-tuple method decomposes a given pattern into several sets of n points, termed n-tuples. The classifier stores class-specific information about the training set in a number of look-up tables or RAM nodes. The entries in each look-up table are addressed by sampling n specific data locations of the input that constitutes a 'feature' of the pattern. A pattern is classified as belonging to the class for which it has the most features in common with at least one training pattern of that class.

Figure 1 shows an n-tuple network, which is built out of RAM nodes. The input address of the RAM unit is also known as "tuple". If the width of the address bus (also known as input connection map) is n bits then the tuple is termed as "n-tuple". The width of the address bus is also known as "tuple-size". Total number of tuples, denoted by R, is the number of tuples available to be optimized. R depends on the network's structure. A group of RAM nodes in a tree-like structure is called a discriminator. The discriminator achieves its goal by presenting to each neuron only a subset of the input pattern, and adding up the outputs of its RAM nodes. This sum can be seen as a measure of the recognition confidence of the discriminator. Therefore, when the discriminator sees a previously learned pattern, its integer output reaches the discriminator's maximum. For an input vector, of size L, the number of necessary RAM nodes R of connectivity n that should be used to cover all inputs of the input vector should satisfy: $R \times n \geq L$. A group of discriminators is used to distinguish a fixed number of classes. The number of classes, which need to be distinguished by a network, determines the number of discriminators needed in a network. The network shown in Figure 1 can be used to distinguish a fixed number of classes. If it consists of 'j' discriminators, it can differentiate j classes. At the output of all discriminators there is a "decision block" where the winner class is chosen using some criteria such as the greatest sum, a threshold of the greatest sum, difference between sums etc. In greatest sum approach, the discriminator containing the greatest number of active RAM nodes is selected. Thus a pattern is 'recognized' as the one whose discriminator 'fired' the most, that is, the discriminator with the highest count of memorized tuples.

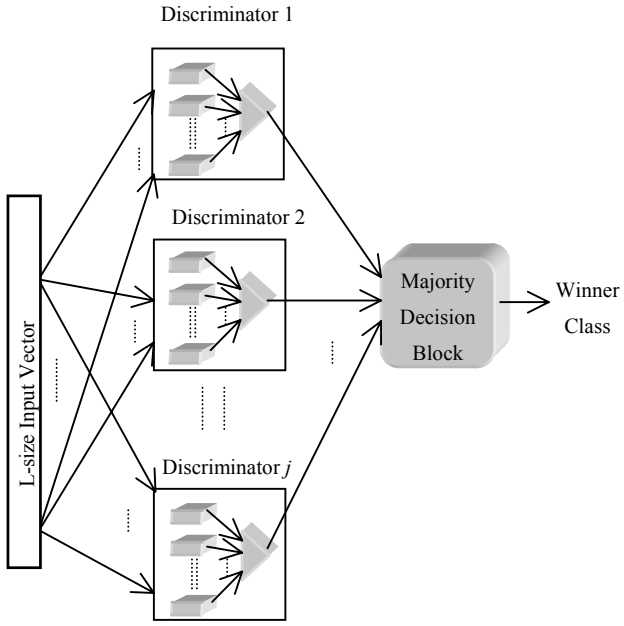

Figure 1. An n-tuple network

The input connection mapping of the n-tuple classifier determines the sampling and defines the locations of the pattern matrix. There will be a vast number of possible connections for a matrix with the dimension like 32 by 32. The classification and generalization performance are highly dependent on these input mappings [5][14]. Conventionally input mappings are randomly chosen [6]. It has been demonstrated in [20] that a randomly connected system perform better than a network with an ordered map. Orderly fashioned input connection failed because the patterns being discriminated were very similar to the way the system was organized. Random connection was favourable because randomness doesn't have a pattern with it. Fairhurst and Stonham [11] have shown that the n-tuple scheme is relatively insensitive to the connection mapping. However, Aleksendar and Stonham [1] have argued that a random map is suitable for an un-optimized problem because sampling points distributed throughout the pattern matrix are more likely to detect global features than an ordered map. For an optimized case better selection of input mappings can give a relatively better performance [1]. Bishop *et al.* [5] demonstrated the importance of sampling sequence in discriminating similar classes.

3. PARTICLE SWARM ON N-TUPLES

When particle swarm optimization is applied to the n-tuple training problem, the "tuples" of the n-tuple can be termed as

"particles". Thus each particle corresponds to an input connection map of the n-tuple network. The size of an n-tuple network is defined by the total number of tuples it is built with. Total number of tuples, denoted by R, is the number of tuples available to be optimized by particle swarm. R depends on the network's structure. The particle swarm technique makes use of a population of particles or input-maps (for n-tuples), where each particle has a position and a velocity. The PSO formulae, as shown in Equation 1 and 2 define each particle as a potential solution in a multi-dimensional search space.

$$V_{i,d}(t+1) = \omega \times V_{i,d}(t) + \psi 1 \times ran1 \times (P_{i,d} - X_{i,d}(t)) + \psi 2 \times ran2 \times (P_{gd} - X_{i,d}(t)) \quad \text{.....(1)}$$

$$X_{i,d}(t+1) = X_{i,d}(t) + V_{i,d}(t+1) \quad \text{.....(2)}$$

The dimension of the PSO corresponds to the bits or the tuple-size of each tuple. As the tuples are "n" bits, so the PSO will be n dimensional with the i-th particle represented as $X_i=(X_{i1},X_{i2},..X_{in})$. The PSO remembers the best position found by any particle which is known as global best, denoted by P_g. Additionally each particle remembers its own previously best found position designated as $P_i=(P_{i1},P_{i2},...P_{in})$ and its velocity $V_i= (V_{i1},V_{i2},...V_{in})$. Equation 1 and 2 will define the velocity and position of the i-th particle with d-th dimension.

Search in PSO starts with the random initialisation of particles' positions and velocities within the allowed range defined by X_{max}, X_{min}, V_{max} and V_{min}. Usually V_{min} is the negative of V_{max}. Each particle keeps track of its own performance. At each iteration, the velocity of every dimension of a particle gets updated according to Equation 1, where $V_{i,d}$, $P_{i,d}$ and P_{gd} constitute the particle's momentum. As this momentum is different for different dimension of a particle, this has effect to force the particle to change the trajectory in the search space towards the most promising areas. This momentum is essential, as it is the feature of PSO that allows particles to escape the local optima. In addition the *ran1* and *ran2* in Equation 1 adds some random adjustments in velocities, which is essential to avoid the situation where the particle endlessly follows the exact same path. Constants $\psi 1$ and $\psi 2$ in Equation 1 determine the relative influence of the "individuality" and "sociality" traits of the particles and are usually both set the same to give each component equal weight as the individual and social learning rate.

3.1 PSO-based tuple search

Tuple search algorithm by PSO is being illustrated in the flow chart in Figure 2. The algorithm starts with Q particles. Q is the total number of particles (population size) in any iteration and they are initially distributed randomly over the whole pattern matrix. The target is to find R class-specific tuples in total. Class-specific tuples best describe a specific class but also describe other classes to some extent [2]. The distribution of R tuples among the classes is proportionate to the error rates [2]. So the class with the most error rate gets the most number of tuples and the class with the least error rate gets the least number of tuples.

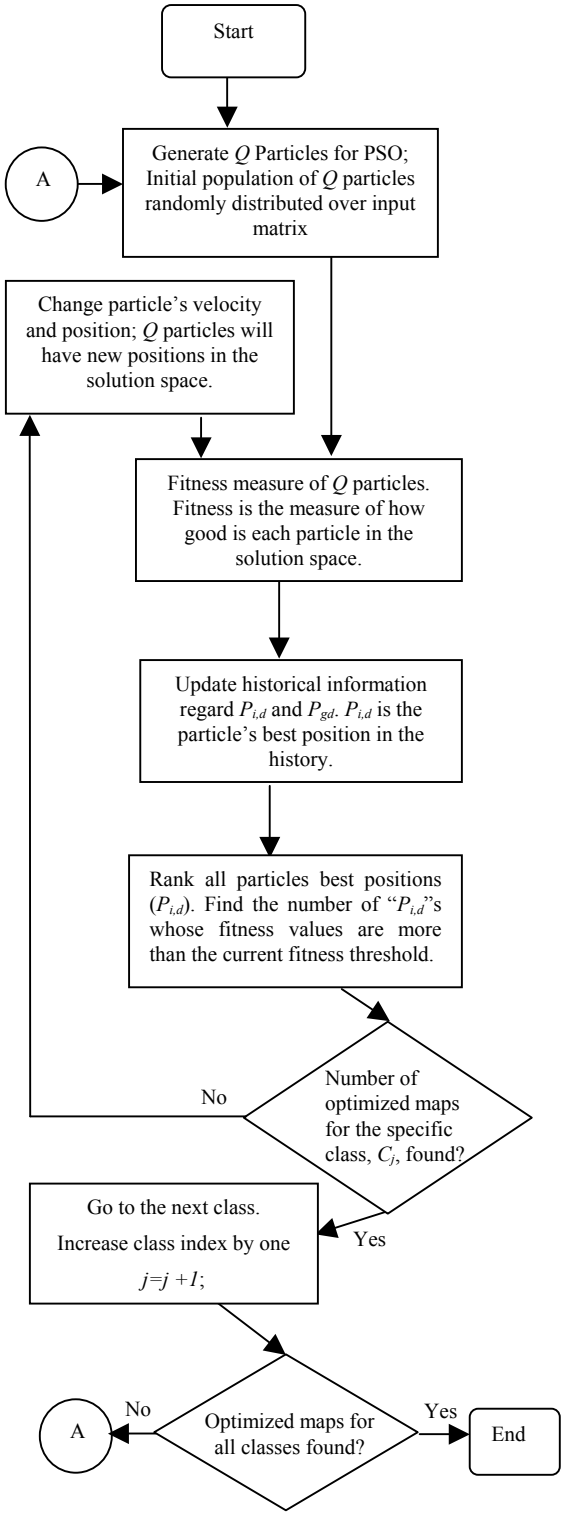

Figure 2. Flow chart of PSO based tuple search

Fitness of each particle is measured according to a reward and punishment based scheme [2], where a reward is associated with the correct recognition of the pattern and the penalties for misclassification and rejection. Based on fitness results each

particle's best positional values are updated. '$P_{i,d}$' defines the location along the dimension *d* of the best positional value of each particle in the history. So '$P_{i,d}$'s represent best positions of all particles so far. Next in the flow chart (Figure 2), fitness of all $P_{i,d}$ particles are compared with a fitness threshold described by Equation 3, where $\max_i(O_{ji}(t))$ is the score of the best-performed tuple among all the tuples in the current iteration.

$$Threshold = \max_i(O_{ji}(t)) \times (1 - exp(-\tau / t)) \quad \text{.....(3)}$$

Fitness threshold exponentially decays over iterations according to the above equation, where τ should be carefully chosen and varied throughout the search as a trade-off between performance and speed. A solution falling within the threshold distance of a specified value would be considered as an acceptable solution. The algorithm checks if the total number of particles for a class has been found. If the target number of particles for a class are not found then the particles velocities and positions will be updated, according to Equation 1 and 2, to explore new locations in the search space. After finding all optimized particles for a class, tuples for the next class group will be sought. At the beginning of searching for the next class a new population of *Q* particles will be reinitialised randomly. Once all the particles are sought for all classes the optimization task will be completed and an optimal set of *R* maps will be found. These *R* maps will be used as input connection maps of the n-tuple network to recognize characters.

Particle swarm optimization like any other stochastic algorithm may prematurely converge [19]. Fast rate of information flow between particles can create similar particles resulting in less diversity in the system, thus increasing the possibility of being trapped in local optima [22]. PSO is also very much problem dependent like any other stochastic search. No single parameter setting exists which can be applied to all problems [19]. For example choosing the value for the inertia weight, ω in Equation 1, could be critical. A large inertia weight favours exploration (global search), while a small inertia weight favours local search [26].

4. SELF-ORGANIZED CRITICALITY

To understand the concept of SOC lets consider a pile of sand. At some point, as grains of sand are slowly and steadily added, the pile becomes "critical" or unstable, and an avalanche occurs spontaneously. In the sand-pile model [4] grains are dropped on a lattice, they can pile up until a specified height is reached, after which they fall on the neighbouring sites. In this way avalanches propagate through the system until they fall out of the boundaries. Now, this visual and obviously simple system is, in fact, complex (there are truly many sand grains interacting), and, as the pile grows, it must attain the point of criticality, which initiates the dramatic reorganization caused by the avalanche. Bak [4] developed a simple mathematical model to simulate a growing sand pile, and it also produced avalanches. The main idea in SOC is that most state transitions in a component of a complex system only affect its neighbourhood, but once in a while entire avalanches of propagating state transitions lead to a major reconfiguration of the system.

5. HYBRID SOC-PSO

Hybridization helps to combat premature convergence in PSO and it refers to combining different approaches to benefit from the advantages of each approach [19]. Hybridization has been successfully applied to PSO by many researchers [10][16][19]. Lovbjerg and Krink [19] have explored extending the PSO with the SOC to improve population diversity. The SOC-PSO algorithms used for the experiments in the research had a globally set "criticality limit", denoted by *CL*, which is the maximum number of times a position in the search space can be considered or taken in forming a particle. If the criticality value of a position in the search space exceeds this limit, the particle corresponding to that position responds by dispersing the criticality within its surrounding neighbourhood and then by relocating itself. Two types of relocation were investigated in [19]: the first reinitialises the particle, while the second pushes the particle with high criticality a little further in the search space. The second approach was followed in the SOC-PSO model used for our research. If the redistribution causes the criticality of the surrounding cell to be increased then process continues until criticalities of all the positions are below the maximum limit. The pseudo code of the SOC-PSO algorithm is given in Figure 3.

```
begin
   initialise
   while(not terminate condition) do
    begin
            run PSO{
               for i=1 to the population size Q,
                    for d=1 to the problem dimensionality n,
                          Apply the velocity update equation;

                          Limit magnitude, V_i,d;
                          Update Position, X_i,d;
                          criticality  [X_i,d]= criticality[X_i,d]+1;
                          while (Criticality value at X_i,d >CL)
                                {criticality[X_i,d] = criticality[X_i,d]-1;
                                 X_i,d =Disperse (X_i,d);
                                 criticality[X_i,d] = criticality[X_i,d]+1;}
                    End-for-d;
                    Compute Fitness;
                    If needed, update historical information
                    regarding P_i,d and P_gd;
               End-for-i;
    End
 End
**********************************
Function Disperse (X_i,d)
   {Xnew= f{ X_i,d, random(0 to 7)};
 return Xnew;}
```

Figure 3. Pseudo code of the SOC-PSO algorithm

5.1 Dispersion of Criticality

From the pseudo code of SOC-PSO it can be seen that the SOC algorithm was implemented within the PSO loop. Once the velocity and a new positional value are found in PSO, the criticality value of the new position is being checked. If the value is more than the criticality limit than the dispersion phenomena was realized and it was implemented by choosing a new location

next to the previously found position. New position's criticality value was checked again and if the value was found to be more than the criticality limit then again the dispersion will occur. Thus the dispersion continues until the system finds a location where the criticality value is less than the limit. The flow chart of the SOC demonstrates this fact in Figure 4.The positions on the search space can be considered as a grid as shown in Figure 5. $X_{i,d}$ in the figure represents a position which was found to have a critical value more than the limit. The dispersion was realized by a random jump from $X_{i,d}$ to one of its surrounding positions. There are eight possible positions to jump around $X_{i,d}$ numbered from 0 to 7. In dispersion one of the values from 0 to 7 was chosen randomly and this value will define the new position. The arrow in Figure 5 shows the direction of jump.

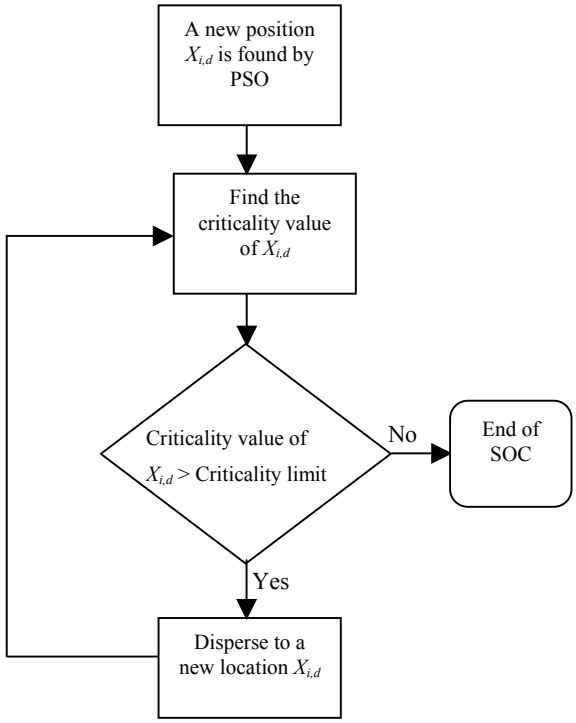

Figure 4. Flow chart of the SOC

5.2 Criticality Limit

Criticality limit (*CL*) can control the diversity of the PSO. A particle will disperse when criticality of any of its dimension will be more than the limit. Criticality limit has to be carefully chosen. An increased criticality limit will allow more particles to be crowded in the same location, thus will make the system less diverse. With a small value of *CL* only fewer particles might bring the system to a critical point and results in more dispersion. Later an experiment will be carried out to count the number of times particles disperse in a search for different values of *CL*. The total number of particles and their dimensionality also play important roles in setting up a value of *CL*. To understand this lets consider a swarm system of 4 particles with dimensionality 8 and a 4 by 4 input matrix where the particles are positioned. Such matrix will have 16 locations. With the dimensionality of 8, each particle will take 8 locations in that matrix. There are 4 particles, so the total places required by all particles are 32. Because there are only 16 positions available in the matrix so to accommodate 32 positions for all 4 particles each position needs to be used at least twice. So the criticality limit for this system has to be at least 2. If the limit is 1 then each position will be used only once, so there will be only 16 positions available and this will not be enough to accommodate 32 positions required by 4 particles. An equation was formulated to find the lowest criticality limit. The smallest criticality limit, denoted by CL_{min}, can be found by Equation 4, where W and H are the width and height of a binary image, Q is the population size and D is the number of dimensions of particles in PSO.

$$CL_{\min} = \left\lceil \frac{Q \times D}{W \times H} \right\rceil \quad \text{.....(4)}$$

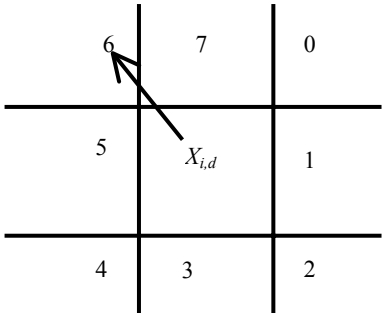

Figure 5. Dispersion by a random jump

6. EXPERIMENTAL RESULTS

Experiments were conducted to search for an optimal set of input connection-maps by the hybrid SOC-PSO algorithm. The NIST [28] database consists of handwritten digits (0,1…9) was used in the experiments. Each character was a binary image with the dimension 32 by 32. All digits were scaled into same dimension and centred. The available tuples were distributed among classes according to the difficulty associated in recognizing the patterns [2]. To calculate the number of class-specific tuples for a class at first the error rate of that class was divided by the total error rate and then the result of the division was multiplied with the total available tuples. The result of the multiplication was rounded to the nearest integer. No normalisation was used in the calculation of class-specific tuples. Providing more tuples to a class with a high error rate ensures that the extra care has been taken for a critical class group. The overall recognition rates, the average of all recognition rates of all classes, were found in the experiment. The recognition rates found by different approaches were mean of ten runs. The network was built out of 150 tuples with tuple-size 8. So total tuples available to be optimized was 150. Because the tuple-size was 8, so the dimensionality of the hybrid PSO algorithm was 8. The task was to use hybrid PSO algorithms to selectively choose tuples that describe the classes better and later use these tuples to recognize a test data set.

Table 1. Results of t-test for SOC-PSO

2nd Algorithm	t-value	p-value
Random selection [6]	15.16	1
Hill-climbing type [2]	5.76	1
Genetic Algorithm [3]	5.57	1
PSO ($\psi 1$=1, $\psi 2$ =1, V_{max}=2)	1.85	0.96

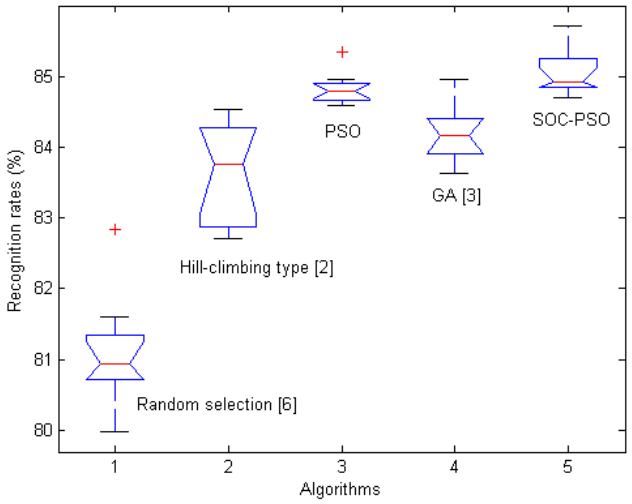

Figure 6. Box plot of training algorithms

Several variations of the SOC-PSO were conducted with different values of the criticality limit, $\psi 1$, $\psi 2$ and particle velocities. The inertia parameter can be decremented with number of iterations from 0.7 to 0.4 as in [19]. In our approach as the SOC-PSO can terminate at any iteration, the inertia parameter was chosen to be a constant value of 0.7. Further diversity was added to the SOC-PSO by reducing the overlapping level, denoted by '*OL*', between any two particles. *OL*=1 means that only the one dimensional value of a particle is allowed to match with any one dimensional value of any other particle. Among various approaches a version of the SOC-PSO (with the settings $\psi 1$=1, $\psi 2$ =1, V_{max}=2, *CL*=2 and *OL*=1) exhibited 4.12% higher recognition rate when compared to a conventionally trained n-tuple network [6]. The improvement by SOC-PSO over a hill-climbing type approach [2] and GA [3] were 1.38% and 0.88% respectively. Statistical significance of the results was analysed by the student's t-test [8]. The best-performed SOC-PSO was compared against other algorithms. The null hypothesis for the test was "average recognition rate by the SOC-PSO is higher than a second algorithm". For 10 trials of each algorithm the degrees of freedom [8] was 18. In the test, t-values were calculated from the experimental results and compared against the theoretical t-values at different confidence levels [8] and 18 degrees of freedom. Theoretical t-values for 90%, 95%, 99% and 99.9% confidence level and 18 degrees of freedom were 1.73, 2.10, 2.88, and 3.92. The t-values found in the experiment against the null hypothesis are presented in Table 1. Results show that the increases in recognition rates by SOC-PSO over conventional random selection, hill-climbing type approach and the GA based method are statistically "very highly significant" [8] because the experimental t-values for all of these cases were greater than the theoretical t-value (3.92) at 18 degrees of freedom. The observed t-value against the original PSO was 1.85 (greater than 1.73) and this implies that the improved results by the SOC-PSO over the classical PSO were statistically significant at 90% confidence level. The p-values in the table indicate the probability of observing the result by chance given that the null hypothesis is true. Small values of probabilities cast doubt on the validity of the null hypothesis.

Table 2. Dispersion for different Criticality set-up

Criticality Limit, *CL*	Dispersion Count (Avg of 30 cycles)
1	∞
2	748
3	111
4	28
5	8
6	2
7	1
>8	0

Figure 6 displays the side-by-side box plots [27] of the results found in the experiments. Each box in the figure was constructed with the recognition rates of ten trials. The box plot conveys location and variation information in data sets, particularly for detecting and illustrating location and variation changes between different data groups of algorithms. The notches in Figure 6 are drawn about the median so that notches that don't overlap represent significant differences between medians (with 95% confidence). The median of recognition rates for SOC-PSO was above 85%, for PSO was just below 85%, for hill-climbing was just below 84%, for GA was just above 84% and for randomly selected approach was near 81%. Clearly the SOC-PSO exhibited a higher median than any other algorithm. Box plots also show if there are unusual observations (outliers) in the dataset. Outliers are individually identified with a plus symbol in Figure 6. Two unusual observations were plotted: one for the random selection and the other one for the PSO.

Table 2 shows dispersion count or number of times particles dispersed for different values of criticality limit. Dispersion count in the table was calculated by finding the average numbers of dispersion in 30 cycles or iterations. Results show that when the criticality limit, *CL*, was equal to or greater than 8 there was no dispersion by any particle. This is because there was no situation where a particle's criticality could cross the limit. Dispersion count was found to be high for a small value of a criticality limit. It showed highest value for a criticality limit of 2 and then the value was gradually dropped to 1 when the criticality limit was 7. A small value in *CL* exhibited better results in the experiments by directing the search to explore new locations. But this benefit was achieved by the system with the expense of spending more time in searching due to dispersion.

Table 3. Dispersion in a typical SOC-PSO cycle for *CL*=5

Particle Position, $X_{i,d}$	Direction of dispersion
107	0
536	2
373	0
107	3
847	7
289	6
579	0
373	3

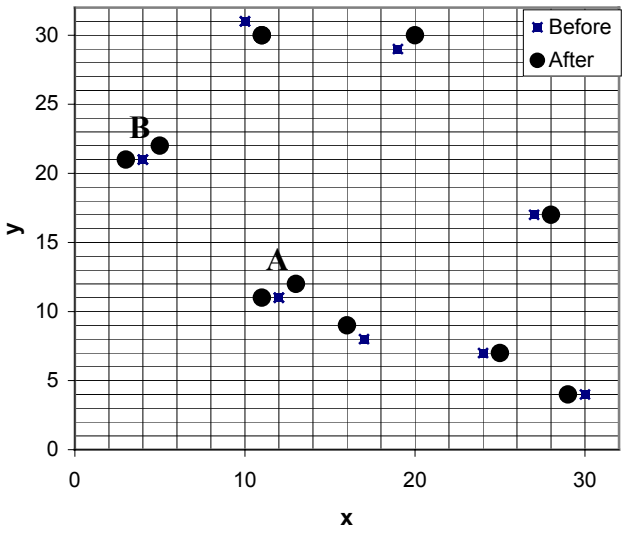

Figure 7. Co-ordinates of $X_{i,d}$ from Table 3

From Table 2 it can be noted that when *CL* was 1, particles dispersed for infinite times or 'forever'. An infinite loop made the system non-convergent and hence was not acceptable. To avoid a situation where the system might fall into an infinite loop, Equation 4 was formulated, which calculates the minimum value of *CL*. In the experiment there were 200 particles (*Q*), each with a dimensionality 8 (*D*). Area of each image was 32 (*W*) by 32 (*H*). Once these values are put in Equation 4, CL_{min} has come out to be 2. Table 3 holds the data of dispersion of particles in a SOC extended PS optimized system. Results in the table were taken from a typical search cycle of a SOC-PSO simulation in training n-tuple classifier for *CL*=5. The first column in the table shows positions of the particles where dispersion occurred and the second column shows the direction of dispersion or direction of jump around $X_{i,d}$ as shown in Figure 5. Figure 7 shows the x-y co-ordinates of $X_{i,d}$ from Table 3 in a 32 by 32 image area. Small squares and circles in the figure depict the positions of the particles before and after dispersion respectively. As dispersion is realized in the nearest neighbourhood area, so a circle in the close proximity of a square would most likely represent the position after dispersion. Position "*A*" in Figure 7 corresponds to a value of 373 of $X_{i,d}$ in Table 3. It can be noted from the table that there are two occasions where the value of $X_{i,d}$ was 373, but for both cases the directions of jump were different and this fact is portrayed by the two circles next to the position *A* in Figure 7. A similar situation was observed next to the position *B* in the figure. Dispersion was most observed when the value of *CL* was 2 (Figure 8). A low value of *CL* forces the system to reach to the criticality point too often and therefore causes more dispersion.

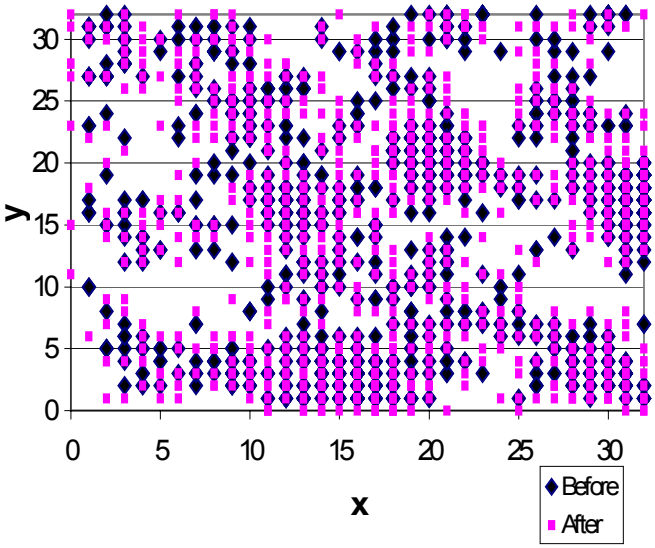

Figure 8. Dispersion in a typical SOC-PSO cycle for *CL*=2

7. CONCLUSIONS

This paper described a novel SOC-PSO hybrid technique and how this approach was applied to optimize an n-tuple network for recognizing binary handwritten characters from the NIST database. The experiments suggest that values of the criticality limit play an important role in the exploration of solutions for a SOC optimized network. A low value in *CL* was preferred to assist exploration for new solutions. It was clear that, by using the hybrid PSO, the connectivity pattern is rearranged in such a way that the more relevant features of the input patterns in the training set, for each class, tend to have more connections to the nodes in the group of tuples specific to that class. This leads to improved performance of the n-tuple classifiers. The observed improvements for the SOC-PSO algorithm were found to be statistically significant when compared to other approaches.

8. REFERENCES

[1] Aleksander, I., and Stonham, T.J. (1979). Guide to Pattern Recognition using Random-access Memories, *Computers and Digital Techniques*, vol. 2, pp. 29-40.

[2] Azhar, M. A. H. B. and Dimond, K.R. (2004). A Stochastic Search Algorithm to Optimize an N-tuple Classifier by Selecting Its Inputs, *International Conference on Image Analysis and Recognition*, Porto, Portugal, Springer-Verlag, September 29 - October 1.

[3] Azhar, M.A.H.B., Deravi, F. and Dimond, K.R.(2005). Relative Performances of Swarm Intelligence and Genetic

Algorithm to Select Better N-tuples, *Proceedings of the IEEE SMC UK-RI Chapter Conference on Applied Cybernetics*, pp. 111-116, London, UK, 7-8 September.

[4] Bak, P. (1996). How nature works: the science of self-organized criticality (*Copernicus*, New York).

[5] Bishop, J.M., Crowe, A.A., Minchinton, P.R. and Mitchell, R.J. (1990). Evolutionary Learning to Optimise Mapping in n-Tuple Networks, *IEE Colloquium on "Machine Learning"*, 28 June, Digest 1990/117.

[6] Bledsoe, W. and Browning, I. (1959). Pattern recognition and reading by machine, *Proceedings of Eastern Joint Computer Conference*, pp. 225-232, Birmingham.

[7] Boettcher, S. and Paczuski, M. (1997). Aging in a Model of Self-Organized Criticality, *Physical Review Letters*, vol.79, Issue 5, pp. 889-892, The American Physical Society.

[8] Deacon, J. (2006). The Really Easy Statistics Site, Biology Teaching Organisation, University of Edinburgh, http://www.biology.ed.ac.uk/research/groups/jdeacon/statistics /tress1.html

[9] Esmin, A. A. A., Aoki, A. R., and Lambert-Torres, G. (2002). Particle swarm optimization for fuzzy membership functions optimization. *Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, pp. 108-113.

[10] Esquivel, S.C. and Coello Coello, C.A.(2003). On the Use of Particle Swarm Optimization with Multimodal Functions. In *Proceedings of the IEEE Transactions on Evolutionary Computation*, vol. 2, pp. 1130-1136.

[11] Fairhurst, M.C. and Stonham, T.J. (1976). A Class. System for Alpha-Numeric Characters Based on Learning Network Techniques, *Digital Processes*, vol. 2, pp. 321-339.

[12] Garcia, L.A.C. and Souto, M. C. P. (2004). Global Optimisation Methods for Choosing the Connectivity Pattern of N-tuple Classifiers, *Proc. of the IEEE International Joint Conference on Neural Networks*, Budapeste, pp. 2263-2266.

[13] Jain, A.K., Duin, R.P.W. and Mao, J. (2000). Statistical pattern recognition: A review, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 4-38

[14] Jørgensen, T. M., Christensen, S. S. and Liisberg, C. (1995). Crossvalidation and information measures for RAM based neural networks, *Proc. of the Weightless Neural Networks Workshop*, University of Kent at Canterbury, UK, pp. 87-92.

[15] Jørgensen, T.M., Linneberg, C. (1999). Theoretical analysis and improved decision criteria for the n-tuple classifier, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, pp.336-347.

[16] Kalyan, V., Thanmaya, P., Chilukuri, K. M. and Lisa, A. O. (2003). Optimization Using Particle Swarms with Near Neighbor Interactions, *GECCO*, July 11-16, Chicago, Illnois.

[17] Kennedy, J., and Eberhart, R. C. (1995). Particle swarm optimisation, *Proc. of the 1995 IEEE Int. Conf. on Neural Networks* (Perth, Australia).

[18] Krink, T. and Thomsen, R.(2001). Self-Organized Criticality and Mass Extinction in Evolutionary Algorithms, *Proceedings of the Third Congress on Evolutionary Computation (CEC-2001),* vol. 2, pp. 1155-1161.

[19] Løvbjerg, M. and Krink, T. (2002). Extending particle swarm optimisers with self-organized criticality, *Proc. of the IEEE Congress on Evolutionary Computation*, Honolulu, Hawaii USA.

[20] Picton, P. (2000). Neural Networks (*Grassroots Series*), 2nd Edition, Palgrave Publishers Ltd.

[21] Rickers, P., Thomsen, R. and Krink, T. (2000). Applying Self-Organized Criticality to the Diffusion Model, Late Breaking Papers at the *2000 Genetic and Evolutionary Computation Conference*, vol. 1, pp. 325-330, Morgan Kaufmann Publishers.

[22] Riget, J. and Vesterstrøm, J.S. (2002). A Diversity-Guided Particle Swarm Optimizer -The ARPSO, *Technical report*, Department of Computer Science, University of Aarhus.

[23] Rohwer, R. and Morciniec, M. (1998). The Theoretical and Experimental Status of the n-tuple Classifier, *Neural Networks*, 11(1): pp. 1-14.

[24] Rohwer, R. and Cressy, D. (1989). Phoneme classification by boolean networks, *Proceedings of the European Conference on Speech Communication and Technology*, pp. 557-560.

[25] Settles, M. and Rylander, B. (2002).Neural network learning using particle swarm optimisers, *Advances in Information Science and Soft Computing,* pp. 224-226. WSEAS Press.

[26] Shi, Y, and Eberhart, R. (1998). Parameter selection in particle swarm optimization, in *Evolutionary Programming VII*, pp. 591-600, Springer, Lecture Notes in Computer Science 1447.

[27] Tukey, J.W. (1977), Exploratory Data Analysis, Reading, MA: Addison-Wesley.

[28] Wilkinson, R., Geist, J., Janet, S., Grother, P., Burges, C., Creecy, R., Hammond, B., Hull, J., Larsen, N., Vogl, T., and Wilson, C. (1992). *The first census optical character recognition systems conference*, Technical Report NISTIR 4912, National Institute of Standards and Technology (NIST), Gaithersburg, USA.