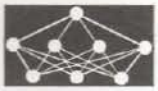

BINARY PARTICLE SWARM OPTIMIZATION IN CLASSIFICATION

*Alejandro Cervantes, Inés Galván, Pedro Isasi**

Abstract: Purpose of this work is to show that the Particle Swarm Optimization Algorithm may improve the results of some well known Machine Learning methods in the resolution of discrete classification problems. A binary version of the PSO algorithm is used to obtain a set of logic rules that map binary masks (that represent the attribute values), to the available classes. This algorithm has been tested both in a single pass mode and in an iterated mode on a well-known set of problems, called the MONKS set, to compare the PSO results against the results reported for that domain by the application of some common Machine Learning algorithms.

Key words: *Evolutionary computation, particle swarm optimization, pattern classification, swarm intelligence*

Received: March 18, 2005
Revised and accepted: June 6, 2005

1. Introduction

The Particle Swarm Optimization (described in [1]) algorithm is motivated by a social analogy. It can be related to the Evolutionary Computation techniques, basically with Genetic Algorithms and Evolutionary Strategies. A good review of PSO fields of application can be found in [2], and a theoretical study of a generalised PSO is in [3].

The algorithm is population-based: a set of potential solutions evolves to approach a convenient solution (or set of solutions) for a problem. Being an optimisation method, the aim is finding the global optimum of a real-valued function (fitness function) defined in a given space (search space).

The social metaphor that led to this algorithm can be summarized as follows: the individuals that are part of a society hold an opinion that is part of a "belief space" (the search space) shared by every possible individual; individuals may modify this "opinion state" based on the knowledge of the way it fits the environment, their previous history of states, and the previous history of states of a set of

*Alejandro Cervantes, Inés Galván, Pedro Isasi
Computer Science Department, Universidad Carlos III de Madrid, Avda. de la Universidad 30, 28911-Leganés, Madrid, Spain. E-mail: alejandro.cervantes@uc3m.es, ines.galvan@uc3m.es, isasi@ia.uc3m.es

other individuals called their neighbourhood, meant to represent the relatives or members of the social network whose opinions are taken into account in order to build oneself opinion.

In PSO terms, each individual is called a "particle", and is subject to a movement in a multidimensional space that represents the belief space. Particles have memory, thus retaining part of their previous state. There is no restriction for particles to share the same point in belief space, but in any case their individuality is preserved. Each particle's movement is the composition of an initial random velocity and two randomly weighted influences: individuality, or tendency to return to the particle's best previous position, and sociality, or tendency to move towards the neighbourhood's best previous position.

The base PSO algorithm uses a real-valued multidimensional space as belief space, and evolves the position of each particle in that space using the Eq.1 and Eq.2:

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}) + c_2 \cdot \psi_2 \cdot (p_{gd}^{t} - x_{id}^{t}) \tag{1}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{2}$$

v_{id}^{t}:	Component in dimension d of the ith-particle velocity in iteration t.
x:	Particle position.
c_1 ,c_2:	Constant weight factors.
p_i:	Best position achieved so long by particle i.
p_g:	Best position found by the neighbours of particle i.
ψ_1 ,ψ_2:	Random factors in the [0,1] interval.
w:	Inertia weight.

A constraint (v_{max}) is imposed on v_{id}^{t} to ensure convergence. Its value is usually kept within the interval $[-x_{id}^{max}, x_{id}^{max}]$, x_{id}^{max} being the maximum value for the particle position [4].

The inertia weight is usually decreased linearly from an initial value close to 1.

The PSO algorithm requires tuning of some parameters: the individual and sociality weights, and the inertia factor. However, both theoretical and empirical studies are available to help in selection of proper values [1][3][4][5].

A binary PSO algorithm has been developed in [1][6]. This version has attracted much lesser attention in previous work. In the binary version, the particle position is not a real value, but either the binary 0 or 1. The logistic function of the particle velocity is used as the probability distribution for the position, that is, the particle position in a dimension is randomly generated using that distribution. The equation that updates the particle position becomes Eq. 3:

$$x_{id}^{t+1} = \begin{cases} 1 & if\ \psi_3 < \frac{1}{1+e^{-v_{id}^{t+1}}} \\ 0 & otherwise \end{cases} \tag{3}$$

other individuals called their neighbourhood, meant to represent the relatives or members of the social network whose opinions are taken into account in order to build oneself opinion.

In PSO terms, each individual is called a "particle", and is subject to a movement in a multidimensional space that represents the belief space. Particles have memory, thus retaining part of their previous state. There is no restriction for particles to share the same point in belief space, but in any case their individuality is preserved. Each particle's movement is the composition of an initial random velocity and two randomly weighted influences: individuality, or tendency to return to the particle's best previous position, and sociality, or tendency to move towards the neighbourhood's best previous position.

The base PSO algorithm uses a real-valued multidimensional space as belief space, and evolves the position of each particle in that space using the Eq.1 and Eq.2:

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}) + c_2 \cdot \psi_2 \cdot (p_{gd}^{t} - x_{id}^{t}) \tag{1}$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{2}$$

v_{id}^{t}:	Component in dimension d of the ith-particle velocity in iteration t.
x:	Particle position.
c_1 ,c_2:	Constant weight factors.
p_i:	Best position achieved so long by particle i.
p_g:	Best position found by the neighbours of particle i.
ψ_1 ,ψ_2:	Random factors in the [0,1] interval.
w:	Inertia weight.

A constraint (v_{max}) is imposed on v_{id}^{t} to ensure convergence. Its value is usually kept within the interval $[-x_{id}^{max}, x_{id}^{max}]$, x_{id}^{max} being the maximum value for the particle position [4].

The inertia weight is usually decreased linearly from an initial value close to 1.

The PSO algorithm requires tuning of some parameters: the individual and sociality weights, and the inertia factor. However, both theoretical and empirical studies are available to help in selection of proper values [1][3][4][5].

A binary PSO algorithm has been developed in [1][6]. This version has attracted much lesser attention in previous work. In the binary version, the particle position is not a real value, but either the binary 0 or 1. The logistic function of the particle velocity is used as the probability distribution for the position, that is, the particle position in a dimension is randomly generated using that distribution. The equation that updates the particle position becomes Eq. 3:

$$x_{id}^{t+1} = \begin{cases} 1 & if\ \psi_3 < \frac{1}{1+e^{-v_{id}^{t+1}}} \\ 0 & otherwise \end{cases} \tag{3}$$

v_{id}^t: Component in dimension d of the ith-particle velocity in iteration t.
x: Particle position.
ψ_3: Random factor in the [0,1] interval.

This means that a binary PSO without individual and social influences ($c_1 = c_2 = 0.0$) would still perform a random search on the space (the position in each dimension would have a 0.5 chance of being a zero or one).

The selection of parameters for the binary version of the PSO algorithm has not been a subject of deep study in known work.

The continuous version of the PSO algorithm is the one that has received greater attention, mainly to resolve optimisation problems and training of Neural Networks [7, 8, 9, 10]. Not much work has been done yet using the binary PSO.

One previous work [11] also addresses classification problems, using a different perspective and coding from the ones used in the present work. In that case each particle encodes a single rule and an iteration process is required to generate the full classifier.

The aim of this work is to evaluate the capacity of the binary PSO algorithm in classification tasks. With this purpose, we have tested the binary PSO in two running modes: the single pass mode, that evolves a set of rules that tries to match the whole training set; and the iterated mode, that performs several cycles with a reduced set of rules, each of which tries to classify the patterns not already matched in the previous cycles. A reference set of problems has been chosen for tests and the results have been compared to those obtained by common classification techniques.

This paper is organised as follows: section 2 describes how the particles in the binary PSO algorithm may encode the solution to a classification problem, and the fitness function that is used to evaluate the solutions; section 3 details the experimental setting and results of experimentation; finally, section 4 discusses our conclusions and future work related to the present study.

2. Binary PSO and classification problems

2.1 Binary encoding

The PSO algorithm will be tested on binary (two classes) classification problems with discrete attributes. The solution will be expressed in terms of rules. The PSO will evolve a set of rules that are able to assign a class from the set 0.1 to a pattern set, part of which is used to train the system (train set). Quality of the solution is evaluated testing the classification that the solution does of set of patterns (test set) not used for the system evolution. Patterns are sets of values for each of the attributes in the domain.

Patterns and rules coding is taken from the GABIL system [12] and summarised below.

A pattern is a binary array with a set of bits for each attribute in the domain, plus one bit for its expected classification. Each attribute contributes a number of bits corresponding to the number of different values that the attribute may take. For each of the attributes, a single bit is set, corresponding to the value of the

attribute for that pattern. The class bit is set to 0 if the pattern expected class is class 0, and 1 if not.

A rule is an array of the same length and structure as patterns. The sets of bits for the attributes are interpreted as a conjunction of conditions over the set of attributes, and will be called "attribute masks". Each condition refers to a single attribute; each bit set for an attribute means that the rule "matches" patterns with the corresponding value for that attribute. Attribute masks may have more than one bit set. Finally, the class bit of the rule is the class that the rule assigns to every pattern it matches.

For instance, if the domain has two attributes (X, Y) and three (V_0^x, V_1^x, V_2^x) and two (V_0^y,V_1^y) values respectively for the attributes, a bit string such as "011 10 1" represents the following rule: If ($X = V_1^x$ or V_2^x) and ($Y = V_0^y$) then class is 1. The representation of this sample is as follows:

$$
\begin{array}{l}
Attributes \in \{X, Y\} \\
Values(X) \in \{v_0^x, v_1^x, v_2^x\} \\
Values(Y) \in \{v_0^y, v_1^y\} \\
Classes = \{0, 1\}
\end{array}
$$

$$
\begin{array}{cccccc}
0 & 1 & 1 & 1 & 0 & 1 \\
v_0^x & v_1^x & v_2^x & v_0^y & v_1^y & 1
\end{array}
$$

$$
Rule: \ (X = v_1^x \vee X = v_2^x) \wedge Y = v_0^y \longrightarrow Class = 1
$$

In [11] an alternative encoding is used, where rules express only one condition for each attribute: the attribute either has a specific value (instead of using a set of possible values) or its value is indifferent to the rule.

2.2 Setting of the binary PSO algorithm

2.2.1 Particle encoding of the solutions and particle evaluation

Each particle in a binary swarm will encode one or more rules. The rules are encoded in order (left to right) and evaluated in that order to provide the class that the particle assigns to a pattern. The first rule (from the left) that matches a pattern assigns the class in its class part to the pattern, and classification stops; if no rule classifies the pattern the pattern can be marked as "unclassified". As an option, an extra bit (that has to be encoded as part of the rule set) might be used to assign a given "default class" to the pattern. This is tested in one of the experiments. If any of the rules in a particle classify the pattern, we say that the particle "matches" the pattern.

As the particle dimension has to be decided at start, this means that the maximum number of different rules in a solution has to be estimated in advance and becomes a new parameter for this method.

2.2.2 Single pass mode or iterated mode

We have tested two different modes on one of the problems. The first one just tries to evolve a set of rules that matches the whole training set.

The second operates with a reduced particle (that may encode only a subset of rules) but executes several training cycles. After each cycle, the best particle in the swarm is saved and the patterns that were classified by that particle are excluded from the training set for the next cycle. This loop continues until either a fixed number of cycles is run, or there are no training patterns left unclassified. The iterated algorithm proceeds as follows:

1. Initialize the training and test set.
2. For each cycle from 1to maxcycles,
   (a) If no unclassified patterns remain, go to 3.
   (b) Run the PSO algorithm.
   (c) Select the best particle and save it.
   (d) Classify the training set according to that particle.
   (e) Remove the patterns already classified from the training set.
3. For each saved particle,
   (a) Classify each unclassified pattern in the test set using the rules in the particle.
4. Evaluate the classification made in 3 (calculate the success percentage).

After the final cycle, the test set is classified using the saved particles, starting with the rules in the first saved particle (from the first cycle), and continuing until a rule matches the pattern (otherwise, it is left unclassified).

2.2.3 Fitness function

The fitness function used to evaluate a particle is a measure of how well the rules in the particle classify the training set. Particle fitness is calculated using Eq. 4 in the single pass mode, and Eq. 5 in the iterated mode.

In the single pass mode, fitness takes into account only the number of patterns that the particle classifies in the right class ("good classifications"). Unclassified patterns are part of the "total patterns" in the denominator, so they are considered as badly classified.

$$Fitness = \frac{Good\ classifications}{Total\ patterns} \tag{4}$$

In the iterated mode, the preferred solutions are those that don't misclassify patterns ("bad classifications"). It is better to leave patterns unclassified, as they can be correctly classified in subsequent cycles. After some empirical testing, the fitness function was changed to Eq. 5 for the iterated mode.

$$Fitness = \frac{Good\ classifications - 2 \cdot Bad\ classification}{Total\ patterns} \tag{5}$$

When performing the evaluation of the test set, Eq. 4 is used in both modes.

2.3 Parameter selection

As said before, the binary swarm has not been given as much attention as the continuous version in previous work. This leads to uncertainty about how the knowledge from the continuous version can be extrapolated to the binary version.

One known difference is that the velocity constraint v_{max} operates in a different way in the binary version. This value gives the minimum probability for each value of the particle position, so if this value is low, it becomes more difficult for the particle to remain in a given position. Being $-v_{max}$ the minimum velocity, this probability may be calculated from Eq. 3:

$$P(x_{id}^{t+1} = 1) \geq \frac{1}{1 + e^{v_{max}}} \tag{6}$$

3. Experimentation

3.1 The Monk's problems

The Monk's set [13] is used to evaluate the binary PSO algorithm described in previous sections. Several techniques used in machine learning were tested on these problems in [13] and those results are used for comparison.

The Monk's problems use a domain with 6 attributes $(A_0, A_1, A_2, A_3, A_4, A_5)$, with respectively $3, 3, 2, 3, 4$ and 2 discrete values (represented by integers), and two classes for classification. There are 432 different possible attribute patterns in the set.

3.1.1 Monk's 1

The first Monk's problem provides a training set of 124 patterns and a test set of 432 patterns (the whole pattern set). Patterns include neither unknown attributes nor noise. The solution to the problem can be expressed as: "Class is 1 if $A_0 = A_1$ or $A_4 = 1$, 0 otherwise".

In our rule syntax this condition may be expressed by the following set of rules:

- $A_4 = 1 \rightarrow$ Class 1
- $A_0 = 0 \wedge A_1 = 0 \rightarrow$ Class 1
- $A_0 = 1 \wedge A_1 = 1 \rightarrow$ Class 1
- $A_0 = 2 \wedge A_1 = 2 \rightarrow$ Class 1
- Extra rules for class 0 or a final rule that matches all the patterns and assigns class 0.

In the rules above, if an attribute is not present, its value is meaningless (the attribute mask for the rules has all its bits set).

3.1.2 Monk's 2

This problem is considered the hardest of the three, being a parity-like classification. The solution may be expressed as follows: "If exactly two of the attributes have its first value, class is one, otherwise 0". This classification has to be expressed as many rules of the form:

- $A_0 = 0 \wedge A_1 = 0 \wedge A_2 = 1 \wedge (A_3 = 1 \vee A_3 = 2) \wedge (A_4 = 1 \vee A_4 = 2) \wedge A_5 = = 1 \rightarrow$ Class 1, etc.

Either their complementary rules for class 0 or a final rule that matches all the patterns and assigns class 0 are also required.

The training set is composed of 169 patterns taken randomly, and the 432 patterns are used as test set.

3.1.3 Monk's 3

This problem is similar in complexity to Monk's 1, but the 122 training patterns (taken randomly) have a 5% of misclassifications. The whole pattern set is used for testing. The intention is to check the algorithm's behaviour in presence of noise. The solution is:

- $A_3 = 0 \wedge A_4 = 2 \rightarrow$ Class 1
- $(A_1 = 0 \vee A_1 = 1) \wedge (A_4 = 0 \vee A_4 = 1 \vee A_4 = 2) \rightarrow$ Class 1

Again, either their complementary rules for class 0 or a final rule that matches all the patterns and assigns class 0 are also required.

3.2 Experimental setting for the single pass mode

A set of six experiments, named E_1 to E_6, has been carried out. The first experiments (E_1, E_2 and E_3) are a base reference for comparison with other algorithms, one for each of the Monk's datasets (Monk's 1, Monk's 2, Monk's 3). E_4, E_5 and E_6 use the Monk's 2 set to perform additional experiments in order to study the effect of changes in the PSO parameters.

E_4 is the same experiment as E_2, using 20 rules and a greater number of iterations. It is performed to check if increasing the particle dimension can improve the results. The solution to the Monk's problem 2 requires at least 15 rules, so E_2 can't really obtain the optimum. In this experiment the particle dimension is enough to hold the global solution, but the algorithm takes much longer and has a much greater space to search.

E_5 is the same experiment as E_4 using a default class bit, as described in previous sections. This is the equivalent to a rule that matches all the patterns (all mask bits are 1), but placed to the right side of the particle (so it is only taken into account when the rest of the rules doesn't match a given pattern). Instead of letting this rule evolve, it can be enforced just by adding a single bit to the particle, so the cost is not great.

E_6 is the same experiment as E_5, but uses a linearly decreasing inertia weight (parameter w). It starts at the value of 1.2 and decreases to 0.

As shown in experiment E_6, the inclusion of the variable inertia weight significantly reduces the number of iterations at which the PSO reaches its best result, but also decreases the success rate.

The last two columns in the table show the number of times the optimum was found for both the training and the test set. The swarm finds the optimum about 50% of the time in E_1, and only 1 time (out of 25) in E_3.

Fig. 1 shows the convergence curve for the best swarm in experiments E_1 and E_3 (E_3 was run again with 5000 iterations to perform this comparison). Over 90% success rate is achieved at about iteration 600 (on average).

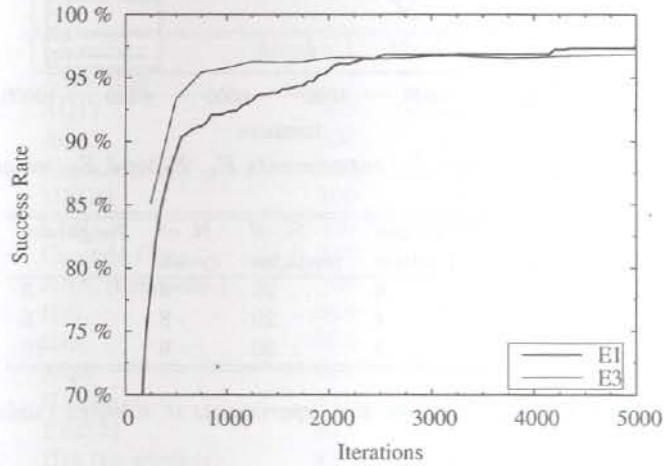

Fig. 1 *Best particle success rate on the test set, experiments E_1 and E_3, single-pass mode.*

The evolution of the success rate in experiments E_4, E_5 and E_6, is compared in Fig. 1. This clearly shows how experiment E_6 (with decreasing inertia weight) converges faster but stays at a lower success rate.

3.4 Experimental setting for the iterated mode

This mode has only been tested on the Monk's problem 2, in an attempt to achieve a more competitive result. We ran 50 simulations for each experiment. For all the experiments in this section, inertia is kept constant and equal to 1.0. The velocity constraint vmax is still removed. From the typical values suggested for the individual and social weights in the literature [1, 3, 4, 5], we have chosen $c_1 = 0.5$, $c_2 = 0.5$, that gave the best results in preliminary testing. The rest of the parameters are summarized in Tab. III.

- Experiment E_7 is the base experiment. We only plan a reduced number of iterations in each cycle to keep this number in the same order of magnitude of the expected number of iterations in the single pass experiments.
- Experiment E_8 tries to find the effect of increasing the number of iterations for E_7.

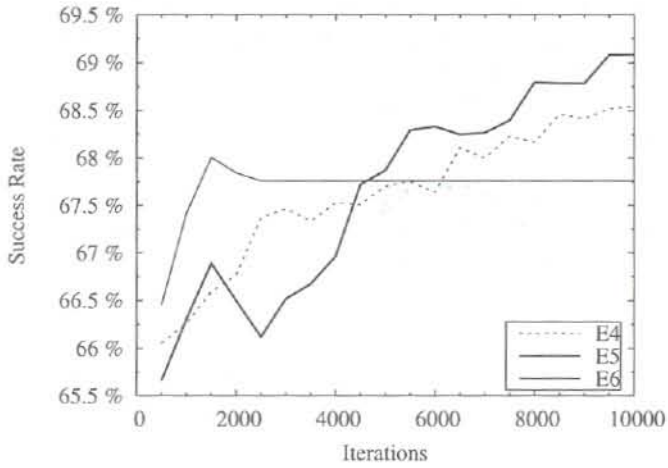

Fig. 2 *Best particle success rate for experiments E_4, E_5 and E_6, single-pass mode.*

Exp.	Max. Iter.	Rules per particle	N. of particles	N. of cycles	Neighbors
E_7	500	4	20	8	5
E_8	1000	4	20	8	5
E_9	500	4	30	8	10

Tab. III *Parameters for the experiments in iterated mode.*

- Experiment E_9 tries to measure the effect of increasing the number of particles.

We must note that the final classifier will have a number of rules equal to the rules in a single particle, times the number of cycles actually performed.

The total maximum number of iterations can be calculated multiplying the maximum number of iterations per cycle times the number of cycles.

However, this mode is much faster in comparison to the single pass mode due to two reasons: particle dimension is a fraction of the former, and each cycle reduces the number of patterns in the training set.

3.5 Experimental results for the iterated mode

Tab. IV summarizes the results for the iterated mode experiments, in terms of average success rate and percentage of instances where a given success rate was achieved. For comparison, the same measures are provided for single pass experiments E_4, E_5 and E_6.

All the experiments show a significant improvement over the single pass mode experiments. Experiments E_7 and E_8 also found good solutions (over 75% success rate) relatively often. We may conclude that 500 iterations per cycle and 20 particles are good enough settings for this problem. Also, increasing the number of particles in E_9 does not necessarily provide better results by itself.

	Exp.	Max. Iter.	Aver. success	Std. Dev.	Better than 70%	Better than 75%	Better than 80%
Iter-	E_7	4000	71.25%	0.042	60%	18%	4%
ated	E_8	8000	70.81%	0.039	62%	14%	-
	E_9	4000	70.38%	0.032	58%	4%	-
Single	E_4	10000	68.44%	0.027	40%	-	-
Pass	E_5	10000	68.88%	0.029	44%	4%	-
	E_6	10000	67.55%	0.027	20%	-	-

Tab. IV *Experimental results in iterated mode and comparison with single pass mode.*

Learning Algorithm	Monk's 1	Monk's 2	Monk's 3
AQ17	100	92.6-100	94.2-100
AQ15	100	86.8	100
Assistant Prof.	100	81.3	100
MFOIL	100	69.2	100
CN2	100	69.0	89.1
Cascade Cor.	100	100	97.2
ANN (backprop.)	100	100	93.1-97.2
ID3	98.6	67.9	94.4
IDL	97.2	66.2	
AQR	95.9	79.7	87
ID5R-hat	90.3	65.7	
PRISM	86.3	72.7	90.3
ID3 (no window)	83.2	69.1	95.6
ECOBWEB	71.8-82.7	67.4-71.3	68.0-68.2
ID5R	79.7-81.7	69.2	95.2
TDIDT	75.7	66.7	
CLASSWEB	63-71.8	57.2-64.8	75.2-85.4

Tab. V *Results of other learning algorithms on the Monk's problems.*

The results show that the iterated mode experiments achieve better average and specific results than the single pass mode experiments, though to compare them properly, some additional experimentation would be useful, trying E_4 to E_6 with more rules per particle (up to 32).

3.6 Analysis of results

The PSO performance can be compared to the results obtained in [13] by several machine learning algorithms shown in Tab. V.

The results above show that the single pass PSO can compete with most of the other algorithms in experiments E_1 and E_3. Both are considered easy problems, as the solution can be expressed with a low number of rules. The mayor problem in that experiments, is that PSO has trouble finding the optimum set of rules. Being E_3 an easier problem than E_1, trouble finding the optimum may be related to the noisy patterns.

Regarding the Monk's 2 problem, the experiments in the single pass mode (E_2, E_4, E_5 and E_6) only show a small increase in success over some of the techniques in Tab. V. The iterated mode experiments (E_7, E_8 and E_9), however, outperform on the average the results of many of those techniques.

Increasing the total number of rules per particle, and the number of iterations, gives better results in the iterated mode, and may do so in the single pass mode too.

In E_5 and E_6, the inclusion of a default class for all unclassified patterns leads to slightly better results. This is used by most of the machine learning methods, but obviously prevents a second run of a refining algorithm over the unclassified patterns as used in the iterated mode. The inclusion of a decreasing inertia weight in E_6 forces the algorithm to converge much faster but may lead prematurely to a suboptimal (even for the algorithm) result.

The algorithm performs fairly well in experiment E_3, where some noise is added to the pattern set. In this case, the PSO improves the results of many other algorithms in Tab. III. Tolerance to noise can be a quite interesting feature of the algorithm.

A main drawback of the algorithm is that there is no way of implementing variable-length particles, which means that the maximum number of rules to use as a solution has to be correctly estimated at the start to fix the particle dimension. This has been a significant problem in the Monk's 2 problem. If the rule number is underestimated, the algorithm will not be able to reach the maximum success rate, as the solution can't be expressed by the rules in a single particle. If the rule number is overestimated, the algorithm is slower, as particle dimension is proportional to number of rules, and the extra dimensionality makes reaching high success rates much harder (it requires a greater number of iterations).

4. Conclusion

The binary PSO algorithm has been tested as a method to resolve classification problems and its results have been compared to those obtained by some popular machine learning techniques in a reference set of problems. The results previously obtained show that the binary PSO algorithm can be competitive with other machine learning techniques, even (and more significantly) in noisy scenarios, although some aspects may be improved. The results for complex problems (Monk's 2) can also improve the results of other methods when an iterated approach is used.

However, the PSO as it stands has problems finding the optimum solution in all cases.

In future work we shall try to characterize the domains that this technique may fit best. Also, a mechanism to implement variable-length solutions will be developed to avoid the requirement to estimate this parameter at the start.

Further work concerning the overall behavior of the binary PSO algorithm, in terms of parameter selection and the usage of inertia weight will also be performed to provide guidance in the selection of proper values for the algorithm parameters.

Acknowledgments

This article has been financed by the Spanish founded research MCyT project TRACER (Ref: TIC2002-04498-C05-04).

References

[1] Kennedy J., Eberhart R. C., Shi Y.: Swarm intelligence. Morgan Kaufmann Publishers, San Francisco, 2001.

[2] Parsopoulos K. E., Vrahatis M. N.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing: an international journal, 1, (2-3), 2002, pp. 235–306.

[3] Clerc M., Kennedy J.: The particle swarm – explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. Evolutionary Computation, **6**, 1, 2002, pp. 58–73.

[4] Shi Y., Eberhart R. C.: Parameter selection in particle swarm optimization. In: Proceedings of the Seventh Annual Conference on Evolutionary Programming, 1998, pp. 591–600.

[5] Shi Y., Eberhart R. C.: Empirical study of particle swarm optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), 1999, pp. 1945–1950.

[6] Kennedy J., Eberhart R. C.: A discrete binary version of the particle swarm algorithm. In: Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, 1997, pp. 4104–4109.

[7] Yoshida H., Kawata K., Fukuyama Y., Takayama S., Nakanishi Y.: A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Transactions on Power Systems, **15**, 4, 2000, pp. 1232–1239.

[8] Zhang C., Shao H.: An ann's evolved by a new evolutionary system and its application. In: Proceedings of the 39th IEEE Conference on Decision and Control, 2000, pp. 3562–3563.

[9] Gudise V. G., Venayagamoorthy G. K.: Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In: Proceedings of the IEEE Swarm Intelligence Symposium (SIS), 2003, pp. 110–117.

[10] Zhang L., Zhou C., Liu X., Ma Z., Ma M., Liang Y.: Solving multi objective optimization problems using particle swarm optimization. In: Proceedings of IEEE Congress on Evolutionary Computation 2003 (CEC 2003), 2003, pp. 2400–2405.

[11] Sousa T., Silva A., Neves A.: Particle swarm based data mining algorithms for classification tasks. Parallel Comput., **30**, 5-6, 2004, pp. 767–783.

[12] De Jong K. A., Spears W. M.: Learning concept classification rules using genetic algorithms. In: Proceedings of the Twelfth International Conference on Artificial Intelligence (IJCAI), **2**, 1991.

[13] Thrun S. B., Bala J., Bloedorn E., Bratko I., Cestnik B., Cheng J., De Jong K., Džeroski S., Fahlman S. E., Fisher D., Hamann R., Kaufman K., Keller S., Kononenko I., Kreuziger J., Michalski R. S., Mitchell T., Pachowicz P., Reich Y., Vafaie H., Van de Welde W., Wenzel W., Wnek J., Zhang J.: The MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Pittsburgh, PA, 1991.