A Comparison between the Pittsburgh and Michigan Approaches for the Binary PSO Algorithm

Alejandro Cervantes
Computer Science Department
Universidad Carlos III de Madrid
Avda. de la Universidad 30,
28911-Leganés, Madrid, Spain.
alejandro.cervantes@uc3m.es

Inés Galván
Computer Science Department
Universidad Carlos III de Madrid
Avda. de la Universidad 30,
28911-Leganés, Madrid, Spain.
ines.galvan@uc3m.es

Pedro Isasi
Computer Science Department
Universidad Carlos III de Madrid
Avda. de la Universidad 30,
28911-Leganés, Madrid, Spain.
pedro.isasi@uc3m.es

Abstract- This paper shows the performance of the Binary PSO Algorithm as a classification system. These systems are classified in two different perspectives: the Pittsburgh and the Michigan approaches. In order to implement the Michigan Approach Binary PSO Algorithm, the standard PSO dynamic equations are modified, introducing a repulsive force to favor particle competition. A dynamic neighborhood, adapted to classification problems, is also defined. Both classifiers are tested using a reference set of problems, where both classifiers achieve better performance than many classification techniques. The Michigan PSO classifier shows clear advantages over the Pittsburgh one both in terms of success rate and speed. The Michigan PSO can also be generalized to the continuous version of the PSO.

1 Introduction

The Particle Swarm Optimization algorithm (described in [1]) is a biologically-inspired algorithm motivated by a social analogy. Sometimes it is related to the Evolutionary Computation (EC) techniques, basically with Genetic Algorithms (GA) and Evolutionary Strategies (ES), but there are significant differences with those techniques.

A review of PSO fields of application can be found in [2]. There are also some good theoretical studies of PSO [3, 4, 5] that address the topics of convergence, trajectory analysis and parameter selection. Also some modifications on the basic algorithm are discussed in [4].

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 three factors:

- the knowledge of the environment (its fitness value)
- their previous history of states
- the previous history of states of their neighborhood.

An individual's neighborhood may be defined in several ways, configuring somehow the "social network" of the individual. Several neighborhood topologies exist (full, ring, star, etc.) depending on whether an individual interacts with all, some, or only one of the rest of the population.

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, the tendency to return to the particle's best previous position, and sociality, the tendency to move towards the neighborhood'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 following equations:

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

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \quad (2)$$

v_{id}^{t}:	Component in dimension d of the i^{th} particle velocity in iteration t.
x_{id}^{t}:	Component in dimension d of the i^{th} particle position in iteration t.
c_1 ,c_2:	Constant weight factors.
p_i:	Best position achieved so long by particle i.
p_g:	Best position found by the neighbors 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 $[-x_{id}^{max}, x_{id}^{max}]$, being x_{id}^{max} the maximum value for the particle position [6].

A large inertia weight (w) favors global search, while a small inertia weight favors local search. If inertia is used, it is sometimes decreased linearly during the iteration of the algorithm, starting at an initial value close to 1 [6, 7].

An alternative formulation of Eq. 1 adds a constriction coefficient that replaces the velocity constraint (v_{max}) [3].

The PSO algorithm requires tuning of some parameters: the individual and sociality weights (c_1 ,c_2), and the inertia factor (w). Both theoretical and empirical studies are available to help in selection of proper values [1, 3, 4, 5, 6, 7].

A binary PSO algorithm has been developed in [1, 8]. This version has attracted much lesser attention in previous work. In the binary version, the particle position is not a

0-7803-9363-5/05/$20.00 ©2005 IEEE.

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 the following:

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

v_{id}^t: Component in dimension d of the i^{th} particle velocity in iteration t.

x_{id}^{t+1}: Component in dimension d of the i^{th} particle position in iteration $t+1$.

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 works.

This binary PSO has not been widely studied and some subjects are still open [9]. Some modifications on the binary algorithm equations propose a quantic approach [10]. A previous article [11] also addresses classification problems. Recently, Clerc [12] proposes and performs an analysis of alternative and more promising binary PSO algorithms.

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 algorithm using the two classical approaches taken from the GA community: the Pittsburgh and the Michigan [13] approaches:

- In the Pittsburgh approach, each particle represents a full solution to the problem; in classification problems, a single particle is able to produce a classification of the data by itself.
- In the Michigan approach, each particle represents part of the solution to the problem. A set of particles or even the whole swarm is used to classify the data.

In this paper we propose the introduction of the Michigan approach PSO classifier to solve certain problems that arise when using the Pittsburgh version: its high dimensionality and its difficulty to represent variable-length solutions.

A reference set of problems (the Monk's set) has been chosen for experimentation, so the results can be compared not only between the two approaches but also with the results of other techniques on these problems [14, 15].

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; section 3 describes how the two approaches (Pittsburgh and Michigan) can be applied to PSO; section 4 includes the modifications made to the original PSO algorithm to implement the Michigan approach; section 5 details the experimental setting and results of experimentation; finally, section 6 discusses our conclusions and future work related to the present study.

2 Binary Encoding for Classification Problems

The PSO algorithm will be tested on binary (two classes) classification problems with discrete attributes. The solution will be expressed in terms of rules. Rule-based classifiers have the advantage that the result is readable, and some knowledge about the reasons behind the classifications may be extracted from the rules.

The PSO will generate a set of rules that 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 set of rules on a different set of patterns (test set) not used for the system training. Patterns are sets of values for each of the attributes in the domain.

Patterns and rules coding is taken from the GABIL system [16], an incremental learning system based on Genetic Algorithms. This coding is described 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. A more compact representation of this sample follows:

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

0	1	1	1	0	1
v_0^x	v_1^x	v_2^x	v_0^y	v_1^y	1

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

3 Approaches for the PSO Algorithm

3.1 The Pittsburgh approach

As said in the Introduction, in this approach, each particle represents a full solution (a classifier).

Thus, each particle in a binary swarm will encode one or more rules. The rules are read from left to right, and evaluated in that order to provide the class that the particle as-

signs to a pattern. The first rule (from the left) that matches a pattern assigns the class that is set in its class part, to the pattern, and classification stops; if no rule classifies the pattern it can either be marked as "unclassified" or assigned a default class.

If some of the rules in a particle classify the pattern, we say that the particle "matches" the pattern.

3.2 The Michigan approach

The former approach has problems for complex domains. The number of rules required to express the solution, which is not known in advance, determines the dimension of the search space. Some estimation has to be made on start and this can't be easily changed while the algorithm is running. That is, the equivalent to variable-length solutions in GA can't be implemented easily.

If dimension is overestimated, the swarm has to update excess dimensions for each particle, with the corresponding loss in performance. In essence, this method will have trouble if it has to scale up to be able to solve complex problems (defined by many rules).

We propose a modification of the PSO algorithm that adopts a collective interpretation of the solution of the problem. This is usually called the "Michigan approach" in the context of GA-based classifier systems.

In this approach, the solution to the posed problem is not a single particle in the swarm, but a subset of its particles. The Michigan approach classifier system was first proposed by Holland, the Learning Classifier System (LCS), in [13]. A more recent example of a classifier system based on LCS is XCS [17].

Many of the classifier systems based on LCS implement a reinforcement learning mechanism. In this case, useful partial classifiers are rewarded in a way that increases their chance to be present in the final solution.

However, in this paper, we use supervised learning. In this case each partial classifier (a single particle) is given a fitness value depending on the comparison between the expected results (that are available for the training set), and the actual results of the classification the classifier makes. In [18], different trends inside the LCS family are studied, and a supervised learning Michigan-style classifier based on GAs (UCS) is proposed.

For the Michigan approach to work in a supervised learning context, the "local" fitness function used to calculate the fitness of a single particle has to be such that favors the global evolution of the system to a good solution for the whole problem.

The basic operation of the PSO algorithm is not modified. Particle dimension depends only on the problem's domain (rule length), while population depends on the number of rules that may possibly conform a solution. On each iteration, the particle positions are updated using the PSO equations. However, the swarm is also evaluated as a whole. In our system, the global evaluation of the swarm is calculated as follows:

1. In each iteration of the PSO algorithm, after particles are updated, we obtain the set of particles that classify the pattern (that is, the set of particles that don't mark the pattern as "unclassified"). This set is called the "matching set" for that pattern.

2. A class is assigned to the pattern using a selection method among the particles in the matching set. In the experiments that follow, the selection method we've used is plain "best particle" selection; that means that the class assigned to the pattern is the class assigned by the best particle (in terms of fitness) in the matching set for that pattern. A voting selection method might also be used. If the matching set for the pattern is void, the pattern is unclassified by the swarm system.

3. Once a class (or no class) is assigned to each pattern in the test set, the swarm may be numerically evaluated with any measure that takes into account the successes and failures in the classification. In our experiments we use Eq. 4, which gives the percentage of good classifications that we use to compare our results with the results of other techniques in [14]. With this equation, unclassified patterns are counted as wrongly classified patterns.

$$Swarm\ Evaluation = \frac{Good\ classifications}{Total\ patterns} \cdot 100 \quad (4)$$

4 PSO Algorithm Modifications for the Michigan Approach

For the success of the Michigan approach, the algorithm has to avoid convergence towards a single high-fitness particle, as this particle can only represent a very limited solution (only one rule).

Given Eq. 1, the sociality term tries to attract particles to the proximity of the good solutions. As a consequence, to change the swarm behavior, in the Michigan approach we dropped the sociality term and introduced the following two modifications in the algorithm. Both have the purpose of maintaining population diversity among particles.

4.1 Competitive Particles

The intention is to enforce competition between particles by making successful particles repel their neighbors. As a particle represents one rule, if a particle with good fitness is already in a given section of the search space, then the rest will explore different parts of that space.

There is previous work concerning repelling particles; for instance, in [19], repulsion is used to increase population diversity inside the standard attractive swarm.

The standard PSO updating equations include an individual and a social factor, that move each particle towards the best positions in its knowledge. To implement competition, the social term in Eq. 1 is replaced by a repulsive term, as shown in Eq. 5 and Eq. 6, that replace the standard PSO equation for the velocity update (Eq. 1).

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}) + c_2 \cdot \psi_2 \cdot d(p_{gd}^{t}, x_{id}^{t}) \quad (5)$$

,where

$$d(p_{gd}^{t}, x_{id}^{t}) = \begin{cases} 1 & if\ p_{gd}^{t} = x_{id}^{t} = 0 \\ -1 & if\ p_{gd}^{t} = x_{id}^{t} = 1 \\ 0 & if\ p_{gd}^{t} \neq x_{id}^{t} \end{cases} \quad (6)$$

Note that, except for the particles positions, Eq. 5 operates with real arguments, while Eq. 3 transforms velocities to binary positions.

The function $d(p_{gd}^{t}, x_{id}^{t})$ ensures that the particle is repelled by the best particle in its neighborhood when their positions are equal in a given dimension. Additionally, this factor is set to 0 if the best particle in the neighborhood is the same particle whose position is being updated.

If the particle (binary) position is 0, the velocity change due to this term will be in the positive direction (for velocities), making the position of the particle more likely to become a 1 when Eq. 3 is applied. The reverse is true when its (binary) position is 1.

The repulsion term is weighted by a random factor (ψ_2) and a fixed weight (c_2) that becomes a new parameter for the algorithm.

The modifications above can be generalized to the continuous version of the PSO by changing the definition of the function $d(p_{gd}^{t}, x_{id}^{t})$ with a distance based on the particles' positions.

4.2 Dynamic Neighborhood

The standard version of PSO uses a fixed neighborhood topology that represents the social influences. Particles are usually organized in a ring (or circle) topology; for N neighbors, neighborhood is usually defined as the particle itself plus the closest (N-1) particles in the ring. We'll use the term "static neighborhood" to refer to this topology.

However, when the repulsion factor is included (Eq. 5), neighborhood is better defined dynamically depending on the proximity of the particles. We'll refer to this topology as "dynamic neighborhood".

For each particle, this dynamic neighborhood is calculated each iteration, based on a proximity criteria. For classification problems, the proximity criteria used is the following: two particles are "close" if they match some common patterns in the training set. This is a phenotype criterium; a correspondent genotype criterium may also be defined (based on binary similarity). The neighborhood index (N) then determines the maximum number of particles selected as neighbors as follows: take the particle itself as the first neighbor, then the particle that shares more patterns as second, and continue in decreasing order of number of patterns shared until N neighbors are selected.

The dynamic neighborhood version requires extra calculations; however, pattern matching for each particle has to be calculated to determine the particle fitness, so these calculations can be cached at some extra space cost to speed up the algorithm.

Suganthan [20] proposed a swarm with a local neighborhood whose size was gradually increased. Brits (in [21]) justifies the introduction of topological neighborhood when searching for multiple solutions in the context of niching techniques, which might be applied to the present work. A different dynamic neighborhood has been proposed in [22] for multiobjective optimisation. The notion of selecting an individual from a neighborhood has also been modified to use the center of clusters of particles in [23].

5 Experimentation

5.1 The Monk's problems

The Monk's set [14] is used to evaluate the binary PSO algorithm described in previous sections. Several techniques used in machine learning were tested on these problems in [14, 15].

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.

5.1.1 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:

- $(A_4 = 1) \rightarrow class1$
- $(A_0 = 0) \wedge (A_1 = 0) \rightarrow class1$
- $(A_0 = 1) \wedge (A_1 = 1) \rightarrow class1$
- $(A_0 = 2) \wedge (A_1 = 2) \rightarrow class1$
- 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 missing, its value is irrelevant; with the selected codification, the rules that represent this situation have all the bits that correspond to those attributes set to 1.

5.1.2 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".

If the encoding allowed the representation of the inequality, the classification might be expressed as several rules (one per combination of two attributes) of the following form:

- $(A_0 = 0) \wedge (A_1 = 0) \wedge (A_2 \neq 0) \wedge (A_3 \neq 0) \wedge (A_4 \neq 0) \wedge (A_5 \neq 0) \rightarrow class1, etc.$

With the encoding we used, rules above have to be changed to express inequality in terms of the values avail-

able to for each attribute as follows:

- $(A_0 = 0) \wedge (A_1 = 0) \wedge (A_2 = 1) \wedge ((A_3 = 1) \vee (A_3 = 2)) \wedge ((A_4 = 1) \vee (A_4 = 2) \vee (A_4 = 3)) \wedge (A_5 = 1) \rightarrow class1$, etc.

Again, 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.

5.1.3 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 behavior in presence of noise. The solution is:

- $(A3 = 0) \wedge (A4 = 2) \rightarrow class1$
- $((A1 = 0) \vee (A1 = 1)) \wedge ((A4 = 0) \vee (A4 = 1) \vee (A4 = 2)) \rightarrow class1$

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

5.2 Experimental Setting

5.2.1 Fitness Function

The fitness function used to evaluate the particles in the Pittsburgh approach is given by Eq. 7:

$$Fitness = \{ \frac{Good}{Total} \tag{7}$$

$Good$: Patterns correctly classified by the particle
$Total$: Number of patterns in the training set

In the Michigan approach, fitness for each particle is calculated using Eq. 8, which takes into account rules that only make good classifications, but also evaluates rules that make bad classifications, with a lower fitness.

$$Fitness = \begin{cases} \frac{Good}{Total} & if\ Bad = 0 \\ \frac{Good - Bad}{Good + Bad} - 1.0 & otherwise \end{cases} \tag{8}$$

$Good$: Patterns correctly classified by the particle
Bad: Patterns wrongly classified by the particle
$Total$: Number of patterns in the training set

5.2.2 Global Swarm Evaluation

The fitness functions defined above are "local" fitness functions, applied only when evaluating an isolated particle. To evaluate the whole swarm as a classifier system, the procedure in section 3.2 is used, where the swarm evaluation is given by Eq. 4.

The system stores the best swarm evaluation obtained and the particles that composed that swarm. This swarm is returned as solution when the maximum number of iterations is reached.

5.2.3 Parameter selection

As it was mentioned before, the binary swarm has not been as widely studied as the continuous version. 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:

$$P(x_{id}^{t+1} = 1) \geq \frac{1}{1 + e^{v_{max}}} \tag{9}$$

To give particles some stability in a given position, the setting for v_{max} should be high, while in the continous (real) PSO it has to be low to ensure convergence.

The inertia weight was fixed to a constant value (1.0) because its influence on the binary PSO is not clear.

We have compared the results of the Pittsburgh and Michigan classifiers on the three Monks problems. Table 1 details the generic parameter values and Table 2 shows the parameter values that vary with the Monk's problem selected.

Approach	Number of experiments	c_1	c_2	v_{max}	w
Pittsburgh	25	2.0	2.0	4.0	1.0
Michigan	50	0.5	0.5	4.0	1.0

Table 1: General Parameters for all the PSO experiments

We executed less experiments in the Pittsburgh approach because of the high number of iterations (and time) they required. In Michigan approach, as we'll see later, iteration time was reduced.

Selection of the weights was made using the typical values proposed in [3, 6], and the repulsion coefficient in the Michigan version was selected after some empirical testing.

Monk's Prob.	Appr.	Exp.	Max. Iter.	Rules / part.	Part.	Neigh.
1	Pitts.	P_1	5000	12	20	5
	Mich.	M_1	1500	1	30	5
2	Pitts.	P_2	10000	20	30	10
	Mich.	M_2	3000	1	60	10
3	Pitts.	P_3	1500	4	20	5
	Mich.	M_3	1500	1	24	5

Table 2: Parameters specific to each Monk's problem and approach (Michigan or Pittsburgh)

In Table 2, column "Rules / part." is the number of rules encoded in a single particle. For the Michigan approach, this number is always 1. For the Pittsburgh approach, this number was an estimation based on the problem's complexity (the number of rules needed to express the solution). The number of particles and neighbors were also selected empirically. For the Michigan approach, the number of particles

has to be enough to represent the number of rules in the solution.

5.3 Experimental Results

Success rate for every experiment is an average of the best result achieved by a particle (in the Pittsburgh approach) or a swarm (in the Michigan approach). That is, the best result is recorded when running an instance of every experiment.

Table 3 reports the average results for the experiments and the percentage of optimum solutions (solutions with 100% accuracy) found. The column "Iter. number" shows the average number of iterations that required the algorithm to reach the given success rate. The column "Rule Eval." is a measure of the number of times the algorithm had to apply a rule to a set of patterns: it is the product of the number of particles, rules per particle and iterations.

Monk's Prob.	Exp.	Iter. number	Rule Eval.	Succ. Rate	Optim. Found
1	P_1	1,561	374,640	96.8%	44.0%
	M_1	464	13,905	97.6%	40.0%
2	P_2	5,398	3,238,800	68.9%	0%
	M_2	1,516	90,936	75.0%	0%
3	P_3	955	76,400	96.9%	4.0%
	M_3	706	16,947	94.1%	0%

Table 3: Results of the experiments for Pittsburgh and Michigan approaches, for each of the Monk's problems

Table 4 shows the minimum and maximum success rate achieved for the Monk's 2 experiment and the distribution of results by ranges.

Exp.	Min. Succ	Max. Succ	> 70%	> 75%	> 80%
P_2	64.1%	75.9%	44.4%	4%	0%
M_2	68.8%	80.8%	94.0%	52%	2%

Table 4: Success rate distribution for the Monk's 2 problem in both approaches, and frequency of results in different success levels

Fig. 1 shows the evolution of the actual (not the best) swarm evaluation at the iteration shown as the horizontal axis, for the Michigan approach. This shows that the whole swarm is converging. Note that this figure is not plotting the same result shown in Table 3, that corresponds to the best solution achieved by the swarm during the iteration of the algorithm.

5.4 Analysis of Results

For reference, the results obtained in [14] by several machine learning algorithms on the Monk's problems are shown in Table 5.

For the Monk's 1 problem, described in [14] as an "easy" problem, both the Pittsburgh and Michigan approaches achieve not only high success rate but they are also able to find the optimum. This is interesting as the lack of ability to find optimum values is one of the binary

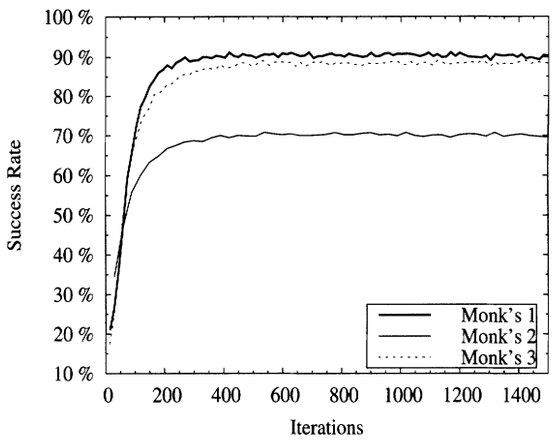

Figure 1: Actual swarm evaluation by iteration number in the Michigan approach, for the Monk's problems 1 to 3 (average of 50 experiments)

PSO drawbacks according to [9]. The Michigan approach achieves a slighter higher success rate and it needs a much smaller number of iterations to reach that success rate.

The motivation of trying the Michigan approach is clearly shown checking the rule evaluations column in Table 3. The Michigan approach outperforms the Pittsburghh approach by an order of magnitude.

For the Monk's 2 problem, which is described as a complex problem, with a XOR-like distribution of patterns, the Michigan approach also clearly achieves a better success rate than the Pittsburgh one. In [14] the results for this problem were in the range 65-70% for most of the classifiers, so the results of PSO in this case are comparable or even better than many of them.

The iteration and number of rule evaluations is quite favorable to the Michigan approach again in this problem. The high evaluation cost for the Pittsburgh approach is due to two reasons:

- the high number of rules needed to represent this problem (high dimensionality).
- the fact that Pittsburgh instances needed a much higher number of iterations in order to reach a better success rate.

Table 4 strongly suggests that the Michigan approach is clearly better than the Pittsburgh approach, because it was able to find classifiers with a success rate better than 70% in the most (94%) of the runs of the Michigan experiment, while Pittsburgh approach is unable to find good solutions in more than half of the runs. This difference is even greater when trying to find more accurate solutions. Pittsburgh approach is almost unable to find solutions with better than 75% success , while Michigan approach reached this value more than 50% of the times.

The Monk's 3 problem, that includes noise, is the one in which the Michigan version is unable to improve the Pittsburgh version. Noise severely degrades the performance of

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

Table 5: Results of other learning algorithms on the Monk's problems, reproduced from [14]

most of the algorithms evaluated in [14], so the overall results are not bad. The Pittsburgh version outperforms many of the classifiers in the Thurn paper. The behavior of the Pittsburgh version in noisy problems could be an interesting feature but needs confirmation using other domains. The Michigan version seems to be more sensible to noise at least in this problem.

All the results above were obtained using the best particle or swarm. The standard PSO equations in the Pittsburgh version should ensure the swarm convergence in many cases. However, this is not proved for the Michigan swarm. Fig. 1 shows the "current success rate" for the three problems. In the presented cases, the swarm has reached the convergence to a certain success rate value.

In order to see the behavior of our approach, a set of rules obtained in experiment M1 (Michigan swarm with the Monk's 1 problem) is shown in Table 6. Only the rules that are selected for classification are shown.

The rules for class 1 can be translated to the exact objective rules in the problem definition:

- $(A_4 = 1) \rightarrow class1$
- $(A_0 = 2) \wedge (A_1 = 2) \rightarrow class1$
- $(A_0 = 1) \wedge (A_1 = 1) \rightarrow class1$
- $(A_0 = 0) \wedge (A_1 = 0) \rightarrow class1$

The swarm has generated both the rules for class 1 and the complementary rules for class 0. This behavior is suboptimal only if rule order is considered. In that case, five rules could be enough: four for one of the classes, and a final "default rule" that assigned all the unclassified patterns to the other class. However, the current rule encoding can't represent rule order: the order in which the rules are evaluated depends only on the fitness function. This idea might be used in further work to reduce the number of particles in the classifier.

A_0	A_1	A_2	A_3	A_4	A_5	Class
100	011	11	111	0111	11	0
111	111	11	111	1000	11	1
110	001	11	111	0111	11	0
011	100	11	111	0111	11	0
101	010	11	111	0111	11	0
001	001	11	111	1111	11	1
010	010	11	111	1111	11	1
100	100	11	111	1111	11	1

Table 6: Sample classifier rules for experiment M_1

6 Conclusions

The binary PSO algorithm has been applied to the resolution of classification problems using a standard rule-based coding. Both a Pittsburgh approach and a Michigan approach were tested with a reference set of problems, the Monk's set. The Michigan approach was introduced because of the dimensionality and lack of flexibility of the Pittsburgh approach for complex problems.

The Michigan approach requires some changes in the PSO algorithm definition, to avoid convergence of the particles and instead force particle diversity:

- Addition of a competitive force that repels a particle from its best neighbor.
- Dynamic neighborhood based on the patterns that are classified by more than one particle (shared patterns).

In the Michigan approach, a very simple evaluation strategy was used to give an global evaluation measure to the performance of the swarm as a classifier.

Results of both methods are good compared to some of the classical classifiers tested over the same problems, although they can't beat the best classifiers in each category.

The results clearly show that the Michigan approach outperforms the Pittsburgh approach except in noisy situations, in which the Michigan version performs slightly worse than the Pittsburgh version. The Michigan approach always requires less iterations to reach its best results. The PSO Michigan classifier is able to provide a good solution with a much lesser number of rule evaluations, and the swarm as a whole converges under the modified PSO movement equations in this paper.

The complexity of the rules obtained by the classifiers is good, given the selected encoding. However, improvement is possible using alternative (rule-based or not) encodings.

Another advantage of the Michigan approach over the Pittsburgh approach is the greater flexibility on the rule composition of the classifier. The number of rules in the solution may be adjusted by varying the number of particles or using only the best N particles in the swarm after the performing evaluation. This means that the complexity of the solution may be easily tuned, and excess rules (particles) pruned. Also a reproduction or extinction schema can be used to adapt the number of particles to the success of the swarm in execution time.

As future work, many improvements from the knowledge in the Machine Learning field can be added to the

Michigan classifier; specifically, a more sophisticated particle evaluation strategy is being investigated.

Also, we are performing more exhaustive investigation on the repulsion term introduced in the algorithm and the dynamic neighborhood topology, that may lead to improvement on the Michigan PSO and deeper understanding of its convergence and stagnation conditions. The modifications made to the PSO equations will also be generalized to the continuous version of the PSO algorithm.

Acknowledgments

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

Bibliography

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

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

[3] Maurice Clerc and James Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evolutionary Computation*, 6(1):58–73, 2002.

[4] Frans van den Bergh. *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, South Africa, 2002.

[5] Ioan Cristian Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Inf. Process. Lett.*, 85(6):317–325, 2003.

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

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

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

[9] Xiaohui Hu, Yuhui Shi, and Russ Eberhart. Recent advances in particle swarm. In *Proceedings of IEEE Congress on Evolutionary Computation 2004 (CEC 2004)*, pages 90–97, 2004.

[10] Shuyuan Yang, Min Wang, and Licheng Jiao. A quantum particle swarm optimization. In *Proceedings of IEEE Congress on Evolutionary Computation 2004 (CEC 2004)*, pages 320–331, 2004.

[11] Tiago Sousa, Arlindo Silva, and Ana Neves. Particle swarm based data mining algorithms for classification tasks. *Parallel Comput.*, 30(5-6):767–783, 2004.

[12] Maurice Clerc. Binary particle swarm optimisers: toolbox, derivations and mathematical insights. http://clerc.maurice.free.fr/pso/binary_pso.

[13] J.H. Holland. Adaptation. *Progress in theoretical biology*, pages 263–293, 1976.

[14] S. B. Thrun et al. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Pittsburgh, PA, 1991.

[15] Shaun Saxon and Alwyn Barry. XCS and the monk's problems. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, page 809, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

[16] Kenneth A. De Jong and William M. Spears. Learning concept classification rules using genetic algorithms. In *Proceedings of the Twelfth International Conference on Artificial Intelligence (IJCAI)*, volume 2, 1991.

[17] Steward W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[18] Ester Bernadó-Mansilla and Josep M. Garrell-Guiu. Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evol. Comput.*, 11(3):209–238, 2003.

[19] T. M. Blackwell and Peter J. Bentley. Dynamic search with charged swarms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2002 (GECCO)*, pages 19–26, 2002.

[20] P. N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1958–1962, 1999.

[21] Riaan Brits. Niching strategies for particle swarm optimization. Master's thesis, University of Pretoria, Pretoria, 2002.

[22] X. Hu and R.C. Eberhart. Multiobjective optimization using dynamic neighborhood particle swarm optimisation. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1677–16, 2002.

[23] J. Kennedy. Stereotyping: improving particle swarm performance with cluster analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1507–1512, 2000.