AMPSO: A New Particle Swarm Method for Nearest Neighborhood Classification

Alejandro Cervantes, Inés María Galván, and Pedro Isasi

***Abstract*—Nearest prototype methods can be quite successful on many pattern classification problems. In these methods, a collection of prototypes has to be found that accurately represents the input patterns. The classifier then assigns classes based on the nearest prototype in this collection. In this paper, we first use the standard particle swarm optimizer (PSO) algorithm to find those prototypes. Second, we present a new algorithm, called adaptive Michigan PSO (AMPSO) in order to reduce the dimension of the search space and provide more flexibility than the former in this application. AMPSO is based on a different approach to particle swarms as each particle in the swarm represents a single prototype in the solution. The swarm does not converge to a single solution; instead, each particle is a local classifier, and the whole swarm is taken as the solution to the problem. It uses modified PSO equations with both particle competition and cooperation and a dynamic neighborhood. As an additional feature, in AMPSO, the number of prototypes represented in the swarm is able to adapt to the problem, increasing as needed the number of prototypes and classes of the prototypes that make the solution to the problem. We compared the results of the standard PSO and AMPSO in several benchmark problems from the University of California, Irvine, data sets and find that AMPSO always found a better solution than the standard PSO. We also found that it was able to improve the results of the Nearest Neighbor classifiers, and it is also competitive with some of the algorithms most commonly used for classification.**

***Index Terms*—Data mining, Nearest Neighbor (NN), particle swarm, pattern classification, swarm intelligence.**

I. Introduction

THE PARTICLE swarm optimizer (PSO) [1] is a biologically inspired algorithm motivated by a social analogy. The algorithm is based on a set of potential solutions which evolves to find the global optimum of a real-valued function (fitness function) defined in a given space (search space). Particles represent the complete solution to the problem and move in the search space using both local information (the particle memory) and neighbor information (the knowledge of neighbor particles).

In this paper, we shall apply both the standard PSO and a novel PSO-based approach in classification problems. A classifier is any system that is able to predict the class to be assigned to a set of data (or patterns); in particular, when the system can use a example set of data (training data) to "learn" how to perform its task, we talk about supervised learning. The classifier must be able to "generalize" from the regularities extracted from data already known and assign the correct classes to new data introduced in the system in the future.

A more specific field in classification is nearest neighbor (NN or 1-NN) classification. NN is a "lazy" learning method because training data is not preprocessed in any way. The class assigned to a pattern is the class of the nearest pattern known to the system, measured in terms of a distance defined on the feature (attribute) space. On this space, each pattern defines a region (called its Voronoi region). When distance is the classical Euclidean distance, Voronoi regions are delimited by linear borders. To improve over 1-NN classification, more than one neighbor may be used to determine the class of a pattern (K-NN) or distances other than the Euclidean may be used.

A further refinement in NN classification is replacing the original training data by a set of prototypes that correctly "represent" it. Once this is done, the resulting classifier assigns classes by calculating distances to the prototypes, not to the original training data, which is discarded. This means that classification of new patterns is performed much faster, as the number of prototypes is much less than the total number of patterns. Besides reducing the complexity of the solution (measured by the number of prototypes), these "Nearest Prototype" algorithms are able to improve the accuracy of the solution of the basic NN classifiers. Note that there are other methods for instance reduction that do not use prototypes but simply choose part of the training set for this task [2], [3]. An evolutionary algorithm approach to the prototype selection problem can be found in [4].

Some work has already been done concerning PSO in classification problems. Most of it concerns rule-based classifiers; for instance, in [5], PSO is used to extract induction rules to classify data; the standard PSO algorithm is run several times, extracting a single rule each time and using only unclassified patterns for subsequent iterations; in [6] and [7], the standard PSO is used for rule extraction in discrete classification problems. There is also some work in fuzzy rule extraction using PSO [8].

Moreover, in [9], a complex hybrid of PSO and Ant Colony Optimization (ACO) is proposed. In this paper, PSO is used both to search for numeric attributes (defining rule clauses based on intervals) and to optimize the pheromone matrix of the ACO algorithm used for nominal attributes.

Finally, in previous work, we used a binary version of PSO to encode induction rules; in [10], sets of induction rules are extracted using an iterated version of the binary version of the PSO algorithm.

Manuscript received April 24, 2008; revised October 6, 2008. First published March 24, 2009; current version published September 16, 2009. This work was supported by the Spanish founded research Project MSTAR::UC3M, Ref: TIN2008-06491-C04-03 and CAM Project CCG06-UC3M/ESP-0774. This paper was recommended by Associate Editor J. Basak.

The authors are with the Department of Computer Science, University Carlos III of Madrid, 28911 Madrid, Spain (e-mail: acervant@inf.uc3m.es; inesmaria.galvan@uc3m.es; pedro.isasi@uc3m.es).

Digital Object Identifier 10.1109/TSMCB.2008.2011816

TABLE I
ENCODING OF A SET OF PROTOTYPES IN A PARTICLE FOR THE PITTSBURGH PSO

	Prototype 1	...	Prototype K	Prototype $K+1$	...	Prototype $N \cdot K$
Position	$X_{1,1}$ $X_{1,2}$ ... $X_{1,D}$	...	$X_{K,1}$ $X_{K,2}$ ... $X_{K,D}$	$X_{K+1,1}$ $X_{K+1,2}$ ... $X_{K+1,D}$	...	$X_{N\cdot K,1}$ $X_{N\cdot K,2}$ ... $X_{N\cdot K,D}$
Class	0	...	$K-1$	0	...	$K-1$

In the context of NN classification with PSO, in [11], the swarm is used to determine the optimum position for the centroids of data clusters that are then assigned the centroid class.

This paper presents two approaches to solve the problem of prototype placement for nearest prototype classifiers.

1) In a standard approach of PSO, a potential solution is encoded in each particle. The information that has to be encoded is the set of prototypes and the prototypes' classes. This approach is tested in this paper and used as reference for the new method proposed later.
2) The second method, called Michigan PSO (MPSO), is still related to the PSO paradigm but uses a Michigan approach; this term is borrowed from the area of genetic classifier systems [12], [13]. To be consistent with the denominations used in that area, the standard PSO is called "Pittsburgh PSO." In the Michigan approach, a member of the population does not encode the whole solution to the problem, but only part of it. The whole swarm is the potential solution to the problem. To implement this behavior, movement and neighborhood rules of the standard PSO are changed. In previous work [14], the authors compared both approaches (Pittsburgh and Michigan) applied to the rule-discovery binary PSO algorithm. The adaptive MPSO (AMPSO) method proposed in this paper is based on the ideas found in [15]. This paper deals with some problems found in the previous work, including a new version of the algorithm with population adaptation, and compares the results with the Pittsburgh approach.

The advantages of the Michigan approach versus the conventional PSO approach are the following: 1) reduced dimension of the search space, as particles encode a single prototype and 2) flexible number of prototypes in the solution.

Moreover, a refinement of MPSO, called AMPSO, is proposed. This version does not use a fixed population of particles; given certain conditions, we allow particles to reproduce to adapt to a situation where a particle of a single class "detects" training patterns of different classes in its Voronoi region.

The way MPSO/AMPSO performs classification may be related to some standard clustering algorithms like Learning Vector Quantization (LVQ) [16] which also search for prototypes that represent known data. However, the way these prototypes are found in MPSO/AMPSO is different in the following ways.

1) Particles use both information from the training patterns and information from the neighbor particles to affect their movement. In LVQ and other methods, prototypes are moved depending only on the position of patterns.
2) Particles use attraction and repulsion rules that include inertia, i.e., velocity retained from previous iterations.
3) We use particle memory (local best position). Even in the absence of outside influence, particles perform a local search around their previous best positions.

In this paper, we were interested in testing MPSO/AMPSO against algorithms of the same family; that is, prototype-based algorithms. Among these algorithms, we have selected 1-NN and 3-NN, LVQ, and Evolutionary Nearest Prototype Classifier (ENPC) [4] for comparison.

For further reference on MPSO/AMPSO properties, we compare MPSO/AMPSO with classification algorithms of different approaches: J48 (tree algorithm, implementation of C4.5, based in trees), PART (rule-based), Naive Bayes, Support Vector Machine (SVM), and Radial Basis Function Neural Network (RBFNN) [17] classifiers, which are successfully applied to several classification problems in [18].

Finally, we also include some algorithms that use an evolutionary approach to extract classification rules, such as GAssist [19] and Fuzzy Rule Learning Algorithm [20].

This paper is organized as follows: Section II shows how the problem is stated in terms of a Pittsburgh PSO; Section III describes the MPSO and AMPSO, including encoding of particles and equations; Section IV describes the experimental setting and results of experimentation; finally, Section V discusses our conclusions and future work related to this paper.

II. Pittsburgh Approach for the Nearest Prototype Classifier

A. Solution Encoding

The PSO algorithm uses a population of particles whose positions encode a complete solution to an optimization problem. The position of each particle in the search space changes depending on the particle's fitness and the fitness of its neighbors.

Data to be classified are a set of patterns, defined by continuous attributes, and the corresponding class, defined by a scalar value. Depending on the problem, attributes may take values in different ranges; however, before classification, we shall scale all the attributes to the [0, 1] range. We are aware that this process may have an effect on the classifier accuracy, so the scaled data sets are the ones used to perform all the experiments.

A prototype is analogous to a pattern, so it is defined by a set of continuous values for the attributes, and a class. As a particle encodes a full solution to the problem, we encode a set of prototypes in each particle. Prototypes are encoded sequentially in the particle, and a separate array determines the class of each prototype. This second array does not evolve, so the class for each prototype is defined by its position inside the particle.

Table I describes the structure of a single particle that can hold N prototypes per class, with D attributes and K classes. For each prototype, classes are encoded as numbers from 0 to $K-1$, and the sequence is repeated until prototype $N \cdot K$. The total dimension of the particle is $N \cdot D \cdot K$.

B. Fitness Function

The Fitness function used to evaluate a particle is simply the classification success rate (1).

To calculate it, first, each pattern in the training data set is assigned the class of the nearest prototype from the prototypes encoded in the particle. If the assigned class matches the expected class of the pattern, we count it as a "Good Classification." The ratio of "Good classifications" to the number of patterns in the data set is the particle's fitness

$$\text{Pittsburgh Fitness} = \frac{\text{Good Classifications}}{\text{Number of patterns}} \cdot 100. \qquad (1)$$

Equation (1) is also used to obtain the overall success rate of the swarm. Once the execution of the algorithm is finished, the prototypes in the best particle obtained are used to classify the validation (test) set. Then, the success rate calculated using (1) over this data set is the result of the algorithm.

C. Algorithm Equations

We shall use the "standard PSO" described in [21], where the authors detail what should be considered a standard version of the algorithm and provide indications about selection of parameters and neighborhood structures.

In brief, PSO uses a real-valued multidimensional space as search space, defines a set of particles located in that space, and evolves the position of each particle using

$$v_{id}^{t+1} = \chi\left(v_{id}^{t} + c_1 \cdot \psi_1 \cdot \left(p_{id}^{t} - x_{id}^{t}\right) + c_2 \cdot \psi_2 \cdot \left(p_{gd}^{t} - x_{id}^{t}\right)\right) \qquad (2)$$

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \qquad (3)$$

where the meanings of symbols are

v_{id}^t	component in dimension d of the ith particle velocity in iteration t;
x_{id}^t	same for the particle position;
c_1, c_2	constant weight factors;
p_i	best position achieved so far 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;
χ	constriction factor.

The neighborhood of the particle may either be composed of the whole swarm (global best, or "gbest" topology) or only a subset of the swarm (local best, or "lbest" topologies). Moreover, some versions of PSO use dynamic neighborhoods, where the relationship between particles changes over time.

III. MPSO Algorithm for the Nearest Prototype Classifier

In the MPSO we propose, each particle represents a potential prototype to be used to classify the patterns using the NN rule. The particle position is interpreted as the position of a single prototype. Movement rules are modified, so instead of finding a best position common to each particle, different particles try to optimize a "local fitness" function that takes into account the performance of the particle in its local environment.

TABLE II
Encoding of a Set of Prototypes in a Whole Swarm in the MPSO

	Position	Class
Particle 1	$X_{1,1}$ $X_{1,2}$ ... $X_{1,D}$	0
Particle 2	$X_{2,1}$ $X_{2,2}$ ... $X_{2,D}$	1
...	...	...
Particle K	$X_{N,1}$ $X_{N,2}$ ... $X_{N,D}$	$K-1$
Particle $K+1$	$X_{N+1,1}$ $X_{N+1,2}$ ... $X_{N+1,D}$	0
...	...	...
Particle $2 \cdot K$	$X_{N+K,1}$ $X_{N+K,2}$ ... $X_{N+K,D}$	$K-1$
...	...	...
Particle $N \cdot K$	$X_{N \cdot K,1}$ $X_{N \cdot K,2}$ ... $X_{N \cdot K,D}$	$K-1$

Each particle has also a class; this class does not evolve following the PSO rules, but remains fixed for each particle since its creation. As classes take part in particle interaction, the swarm is no longer homogeneous, and particles may be considered divided in several types or species.

In the sections that follow, we describe this encoding and the pseudocode for MPSO. Some concepts that differ over the standard PSO are introduced in the pseudocode and are explained in detail in the referenced sections.

The basic variations in equations are the introduction of a repulsion force and the use of a dynamic definition of the neighborhood of a particle. When moving, each particle selects another one from what we call a "noncompeting" set as a leader for attraction, and a second one from a "competing" set as a leader for repulsion. Both neighborhoods are defined dynamically on each iteration and take into account the particles' classes.

As an improvement of the basic algorithm, we finally introduce the possibility of particle creation. We call this version AMPSO. Such mechanism permits the swarm population to adapt to the problem: It is able to grow the number of possible prototypes to use and to modify the class distribution of the prototypes. Whenever a particle detects that it is the closest to patterns of different classes, it has a chance to spawn particles of the required classes. New particles start in the same position as the original particle but move independently from that point on.

A. Solution Encoding

As previously stated, each particle encodes a single prototype, and as such, the dimension of the particles is equal to the number of attributes of the problem. A particle classifies a pattern when it is the closest particle (in terms of Euclidean distance) to that pattern.

Besides its position, each particle is assigned a class. Class is an attribute of the particle and does not change during the iteration of the algorithms. In MPSO, the swarm has to be initialized with enough particles of each class to represent the problem; while in the adaptive version (AMPSO), besides initialization, new particles may appear, and their classes are determined by the patterns they must classify.

Table II represents the structure of a swarm with $N \cdot K$ particles in a problem with D attributes and K classes. Each particle corresponds to a single prototype, so the swarm contains N prototypes per class in the problem. Classes are assigned from 0 to $K-1$ to particles, and the sequence is repeated until the last prototype $(N \cdot K)$. The dimension of each particle is D.

If Tables I and II are compared, we see that the structure of a single particle in the Pittsburgh PSO is analogous to the structure of a whole MPSO swarm.

B. Algorithm Pseudocode and Movement

Our algorithm is based on the PSO algorithm but performs some extra calculations and has an extra cleaning phase. Our additions are explained in the following sections. The overall procedure follows.

```
1) Load training patterns.
2) Initialize swarm; dimension of particles equals number of attributes.
3) Insert $N$ particles of each class in the training patterns.
4) Until maximum number of iterations reached or success rate is 100%:
   a) Check for particle reproduction (AMPSO only), (see Section III-G).
   b) Calculate which particles are in the competing and noncompeting sets of particles for every class (see Section III-E).
   c) For each particle,
      i) Calculate Local Fitness, (see Section III-D).
      ii) Calculate Social Adaptability Factor, (see Section III-F).
      iii) Find the closest particle in the noncompeting set for the particle class (attraction center), (see Section III-E).
      iv) Find the closest particle in the competing set for the particle class (repulsion center), (see Section III-E).
      v) Calculate the particle's next position using (4) and (5) in Section III-C.
   d) Move the particles.
   e) Assign classes to the patterns in the training set using the nearest particle.
   f) Evaluate the swarm classification success using (1).
   g) If the swarm gives the best success so far, record the particles' current positions as "current best swarm."
5) Delete, from the best swarm found so far, the particles that can be removed without a reduction in the classification success value.
6) Evaluate the swarm classification success over the validation set and report result using (1).
```

In step 5) of the previous procedure, a reduction algorithm is applied after the swarm reaches its maximum number of iterations. Its purpose is to delete unused particles from the solution. Particles are removed one at a time, starting with the one with the worst local fitness value, only if this action does not reduce the swarm classification success rating over the training set. The "clean" solution is then evaluated using the validation set.

C. Movement Equations

In MPSO, the equation that determines the velocity at each iteration becomes

$$\begin{aligned} v_{id}^{t+1} = \chi \big(& w \cdot v_{id}^t + c_1 \cdot \psi_1 \cdot \left(p_{id}^t - x_{id}^t\right) \\ & + c_2 \cdot \psi_2 \cdot \operatorname{sign}\left(a_{id}^t - x_{id}^t\right) \cdot Sf_i \\ & + c_3 \cdot \psi_3 \cdot \operatorname{sign}\left(x_{id}^t - r_{id}^t\right) \cdot Sf_i \big) \end{aligned} \tag{4}$$

where the meanings of symbols are

v_{id}^t	component d of the ith particle velocity in iteration t;
x_{id}^t	same for the particle position;
c_1,c_2, c_3	constant weight factors;
p_i	best position achieved by particle i;
ψ_1, ψ_2, ψ_3	random factors in the [0, 1] interval;
w	inertia weight;
χ	constriction factor;
a_i	attraction center for particle i;
r_i	repulsion center for particle i;
Sf_i	social adaptability factor;
sign()	sign function, determines "direction" of influence.

This expression allows the particle velocity to be updated depending on four different influences.

1) The current velocity of the particle, which is retained from iteration to iteration, called "inertia term," same as in the standard PSO: $w \cdot v_{id}^t$.
2) The particle's memory of its previous best position, called "individual term," same as in the standard PSO: $c_1 \cdot \psi_1 \cdot (p_{id}^t - x_{id}^t)$.
3) The current position of a particle that collaborates by exerting an attraction force, called "attraction term": $c_2 \cdot \psi_2 \cdot \operatorname{sign}(a_{id}^t - x_{id}^t) \cdot Sf_i$.

   This term is different from the standard PSO attraction term. The sign() function is used because the magnitude of the attraction does not depend on the distance between the two positions a_{id}^t and x_{id}^t. If the position of the attraction center a_{id}^t is greater than the position of the particle x_{id}^t (to its right), then the velocity component is positive (to the right).
4) The current position of a particle that competes by exerting a repulsion force, called "repulsion term": $c_3 \cdot \psi_3 \cdot \operatorname{sign}(x_{id}^t - r_{id}^t) \cdot Sf_i$.

   This term has no analog in the standard PSO. The sign() function is again used to determine the direction of the velocity change. If the position of the repulsion center r_{id}^t is greater than the position of the particle x_{id}^t (to its right), then the velocity component is negative (to the left).

In the particular case that particle a_i or r_i does not exist, the respective term (attraction term or repulsion term) is ignored.

After velocity update, the particle position is calculated using (5), which is the same equation used in the standard PSO.

$$x_{id}^{t+1} = x_{id}^t + v_{id}^{t+1}. \tag{5}$$

D. Local Fitness Function

In the Michigan approach, each particle has a Local Fitness value that measures its performance as a local classifier. This is the fitness value that is used during the evolution of the swarm to record the best position of the particle.

For this purpose, the algorithm determines the set of patterns to which the particle is the closest in the swarm. We assign the class of the particle to those patterns and determine whether the class matches the expected class for the pattern ("good classification") or not ("bad classification"). Then, we calculate two factors using

$$G_f = \sum_{j \in \{g\}} \frac{1}{d_{ij} + 1.0} \tag{6}$$

$$B_f = \sum_{j \in \{b\}} \frac{1}{d_{ij} + 1.0} \tag{7}$$

where

$\{g\}$	patterns correctly classified by particle i;
$\{b\}$	patterns incorrectly classified by the particle i;
d_{ij}	distance between particle i and pattern j.

In both (6) and (7), we include the distance to the prototypes, so closer patterns have greater influence in the calculation of those factors.

Then, we use (8) to obtain the Local Fitness value for the particle. In this formula, *Total* is the number of patterns in the training set

$$\text{Local Fitness} = \begin{cases} \frac{G_f}{Total} + 2.0, & \text{if } \{g\} \neq \emptyset \\ & \text{and } \{b\} = \emptyset \\ \frac{G_f - B_f}{G_f + B_f} + 1.0, & \text{if } \{b\} \neq \emptyset \\ 0, & \text{if } \{g\} = \{b\} = \emptyset. \end{cases} \tag{8}$$

This fitness function gives higher values (greater than +2.0) to the particles that have only "good classifications," and assigns values in the range [0.0, +2.0] to particles that classify any pattern of a wrong class.

In the lowest range, the particles only take into account local information (the proportion of good to bad classifications made by itself). In the highest range, the particle fitness uses some global information (the total number of patterns to be classified), to be able to rank the fitness of particles with a 100% accuracy (particles for which $\{b\} = \emptyset$).

Note that this function is not used to evaluate the whole swarm; instead, whenever a Michigan swarm has to be evaluated, we calculate the success rate (1), like in the Pittsburgh PSO, but using each particle as a prototype. There are more sophisticated functions in the literature that may be used to evaluate local fitness, to take into account the actual distribution of classes. Experimentation with other fitness functions is desirable and will be subject for future work.

E. Neighborhood for the MPSO

One of the basic features of the Michigan swarm is that particles do not converge to a single point in the search space. To ensure this behavior, interaction among particles is local, i.e., only particles that are close in the search space are able to interact. This means that neighborhood is calculated dynamically using the positions of the particles at each iteration.

Two different neighborhoods are defined for each particle at each iteration of the algorithm.

1) For each particle of class C_i, noncompeting particles are all the particles of classes $C_j \neq C_i$ that currently classify at least one pattern of class C_i.
2) For each particle of class C_i, competing particles are all the particles of class C_i that currently classify at least one pattern of that class (C_i).

When the movement for each particle is calculated, that particle is both:

1) Attracted by the closest (in terms of Euclidean distance) noncompeting particle in the swarm, which becomes the "attraction center" (a_i) for the movement. In this way, noncompeting particles guide the search for patterns of a different class.
2) Repelled by the closest competing particle in the swarm, which becomes the "repulsion center" (r_i) for the movement. In this way, competing particles retain diversity and push each other to find new patterns of their class in different areas of the search space.

Other authors have already used the idea of repulsion in PSO in different ways. For instance, in [22], repulsion is used to avoid a complete convergence of the swarm, and, in [23], increase population diversity in the standard PSO. This allows the swarm to dynamically adapt to changes in the objective function.

F. Social Adaptability Factor

The social part of the algorithm (influence from neighbors) determines that particles are constantly moving toward their noncompeting neighbor and far from their competing neighbor. However, particles that are already located in the proximity of a good position for a prototype should rather try to improve their position and should possibly avoid the influence of neighbors.

To implement this effect, we have generalized the influence of fitness in the sociality terms by introducing a new term in the MPSO equations, called "Social Adaptability Factor" (S_f), that depends inversely on the "Best Local Fitness" of the particle. In particular, we have chosen plainly the expression in

$$Sf_i = 1/(\text{Best Local Fitness}_i + 1.0). \tag{9}$$

G. Adaptive Population

With a fixed population of particles, the MPSO is limited in terms of representation of solutions. It can only find a solution with a maximum number of prototypes. To prevent this limitation, an improved version of MPSO is developed, called AMPSO. AMPSO adjusts the number of particles and their classes to fit the particular problem.

In AMPSO, each particle that classifies a set of patterns of several classes has a probability to give birth to one particle for each of classes in that set. This feature must be used with caution; there is a risk of a population explosion if the reproduction

TABLE III
PROBLEMS USED IN THE EXPERIMENTS

Name	Inst.	Attrbs.	Classes	Class Distribution
Balance Scale	625	4	3	288 / 49 / 288
Bupa	345	6	2	200 / 145
Diabetes	768	8	2	500 / 268
Glass	214	9	6	70 / 76 / 17 / 13 / 9 / 29
Iris	150	4	3	50 / 50 / 50
Thyroid (new)	215	5	3	150 / 35 / 30
Wisconsin	699	6	2	458 / 241

rate is too high and that would worsen the computational cost of the algorithm.

For each particle, we calculate a probability of "reproduction" (P_{rep}) using (12). We decided to give a higher reproduction rate to particles that have a high local fitness value but still classify some patterns of different classes. Therefore, we introduced the best local fitness of the particle in the equation, scaled to the interval [0, 1].

We also introduced a parameter (p_r) in order to tune the probability of reproduction. Finally, we make P_{rep} maximum at the start of the swarm iteration (to improve exploration), and we decrease it lineally until its minimum when the maximum iteration is reached.

New particles are placed in the best position of the "parent" particle, and their velocities are randomized

$$F_{\text{norm}} = \frac{\text{Best Local Fitness} - \text{Min}_{\text{fit}}}{\text{Max}_{\text{fit}} - \text{Min}_{\text{fit}}} \qquad (10)$$

$$It_{\text{norm}} = 1.0 - \frac{\text{Current Iteration}}{\text{Maximum Iterations}} \qquad (11)$$

$$P_{\text{rep}} = F_{\text{norm}} \times It_{\text{norm}} \times p_r \qquad (12)$$

where

Min_{fit}	minimum value for the local fitness function;
Max_{fit}	maximum value for the local fitness function.

IV. EXPERIMENTATION

A. Problem's Description

We perform experimentation on the problems summarized in Table III. They are well-known real problems taken from the University of California, Irvine, collection, used for comparison with other classification algorithms. All the problems have real-valued attributes, so no transformation was done on data besides scaling to the [0, 1] interval.

We have selected problems with different number of classes and attributes. We also include both balanced and unbalanced problems in terms of class frequency. These problems include some that can be efficiently solved by NN classifiers and others in which the performance of NN classifiers may still be improved.

For each problem and algorithm, we performed ten runs with tenfold cross validation, which gives a total of 100 runs over each.

TABLE IV
PARAMETERS USED IN THE EXPERIMENTS

Algorithm	χ	w	c_1	c_2	c_3	p_r
Std. PSO	0.72984	-	2.05	2.05	-	-
MPSO	0.5	0.1	1.00	1.00	0.25	-
AMPSO	0.5	0.1	1.00	1.00	0.25	0.1

B. Parameter Selection

In all the Pittsburgh experiments, we used ten prototypes per class per particle, and 20 particles as population. We used the parameters suggested in [21]. Neighborhood was "lbest" with three neighbors. With these parameters, particle dimension becomes quite high for some of the problems; in this approach, dimension is equal to the number of attributes times the number of classes times ten (from 120 to 540 depending on the problem).

In Michigan experiments, we used ten particles per class for the initial swarm. Particle dimension is equal to the number of attributes (from four to nine depending on the problem), while the swarm initial population ranges from 20 to 60 particles.

The number of iterations was set to 300 both for the Pittsburgh and Michigan experiments, after checking that number was roughly equal to double the average iteration in which the best result was achieved.

In order to compare computational costs, note that, for the given parameters, each iteration in the Pittsburgh approach requires 20 times (the Pittsburgh population size) the number of distance evaluations than an iteration in the Michigan approach.

The values of the swarm parameters for MPSO and AMPSO were selected after some preliminary experimentation. This showed that is was better to use a small value for the inertia coefficient ($w = 0.1$). In all cases, velocity was clamped to the interval $[-1.0, +1.0]$.

Table IV summarizes the values for the rest of the parameters.

C. Experimental Results

In this section, we describe the results of the experiments and perform comparisons between the Pittsburgh PSO and both versions of the MPSO: MPSO, with fixed population, and AMPSO, with adaptive population.

We always use two tailed t-tests with $\alpha = 0.05$ to determine the significance of the comparisons. When we present the results with significance tests, all the algorithms were compared with the algorithm placed in the first column.

In all tables in this section, we use the following notation: a "(+)" tag next to the result of an algorithm means that the average result was significantly better than the result in the first column; "(=)" indicates that the difference was not significant; and "(−)" means that the result was significantly worse when compared to the algorithm in the first column. We also use boldface style to highlight the best result. When differences are not significant, several algorithms may be marked as providing the best result.

In Table V, we compare the average success rate of the Pittsburgh PSO and MPSO. The results show that MPSO achieves a better success rate than the Pittsburgh version except

TABLE V
AVERAGE SUCCESS RATE (IN PERCENT), COMPARISON BETWEEN PITTSBURGH PSO AND MPSO

Problem	Pittsburgh PSO	MPSO
Balance Scale	62.79	**82.59** (+)
Bupa	**65.13**	**64.76** (=)
Diabetes	**74.54**	**74.25** (=)
Glass	74.34	**86.27** (+)
Iris	90.89	**96.70** (+)
Thyroid	88.93	**95.90** (+)
Wisconsin	94.43	**96.50** (+)

TABLE VI
AVERAGE SUCCESS RATE (IN PERCENT), COMPARISON BETWEEN MPSO AND AMPSO

Problem	MPSO	AMPSO
Balance Scale	82.59 (-)	**85.64** (+)
Bupa	**64.76** (=)	**65.25** (=)
Diabetes	74.25 (-)	**75.05** (+)
Glass	**86.27** (=)	**86.94** (=)
Iris	**96.70** (=)	**96.89** (=)
Thyroid	**95.90** (=)	**96.28** (=)
Wisconsin	**96.50** (=)	**96.51** (=)

TABLE VII
AVERAGE NUMBER OF PROTOTYPES IN THE SOLUTION FOR THE THREE ALGORITHMS

Problem	Pittsburgh	MPSO	AMPSO
Balance Scale	26.71	12.90	63.54
Bupa	8.48	11.75	15.04
Diabetes	13.00	11.28	18.49
Glass	10.44	15.58	19.12
Iris	9.82	16.28	17.65
Thyroid	7.19	16.33	19.66
Wisconsin	16.17	11.99	20.98

for the Diabetes and Bupa data sets, where the differences are not significant. Except for those two problems, performance of the Pittsburgh approach was indeed poor, as shown later when comparing Pittsburgh with other algorithms.

In Table VI, we compare the average success rate of MPSO with the success rate of AMPSO, which includes particle creation. It shows that AMPSO is better than MPSO in the Diabetes problem but difference is much more significant in the Balance Scale problem. As we can see in Table VII, the increase in performance is directly related to AMPSO using a much larger number of prototypes in the solution. It seems that the original configuration in Pittsburgh PSO and MPSO (ten particles per class) is unable to represent an accurate solution to that problem. For the other problems, the original choice on initialization seems enough for plain MPSO to provide a good result, as the average success rate of AMPSO is not significantly greater than MPSO.

In Table VII, we show the number of prototypes used in the solution for each problem and algorithm. Even with a fixed population, in this value we only take into account prototypes that are actually used in the solution to classify at least one pattern. The rest are not considered in the solution, so the average number of prototypes in the table is less than the maximum possible (ten prototypes per class in current experiments).

On the other hand, AMPSO allows the swarm population to adapt to the problem so on average it provides solutions with a larger number of prototypes.

The increase in number of prototypes in the solution in AMPSO is larger for the Balance Scale problem (up to 63 prototypes) and the Diabetes problem. In both cases, AMPSO obtained better results than MPSO.

However, this was not the case in the Wisconsin problem, where an important increase in the number of prototypes did not lead to better results. As the result for this problem is already better than the result of basic NN, it may happen that NN classification cannot be improved much beyond that limit without the application of other techniques.

In Table VIII, we show the average number of prototype evaluations needed to reach the solution of each experiment for each of the algorithms. The purpose of this comparison is only to show that MPSO/AMPSO achieve their result with a lower computational cost that the equivalent Pittsburgh approach, when the number of distance evaluations is considered. This factor is calculated by adding to a counter, each iteration, the number of prototypes in the whole swarm on that iteration. When the best solution is recorded for a experiment, we also record the value of this counter.

TABLE VIII
AVERAGE NUMBER OF EVALUATIONS REQUIRED TO REACH THE SOLUTION

Problem	Pittsburgh	MPSO	AMPSO
Balance Scale	99252	5783	24985.71
Bupa	64180	2827	3667.48
Diabetes	64636	3304.4	4968.05
Glass	185412	8596.2	13436.97
Iris	3026.2	4995	5640.25
Thyroid	103170	4955	5985.18
Wisconsin	65596	3575.6	5806.96

For MPSO and AMPSO, the number of evaluations is similar in order of magnitude. The AMPSO needed more evaluations due to the dynamic creation of particles. However, both versions of the MPSO use less evaluations than the Pittsburgh PSO for each of the problems, except for the Iris problem. In the Iris data set, it seems that the Pittsburgh PSO is stuck in a local minimum, as the result in terms of accuracy is poor.

In other problems, values for the Pittsburgh PSO experiments are significantly greater because each of the particles in the Pittsburgh swarm encodes the same number of prototypes than the whole equivalent Michigan swarm. That is, if the Pittsburgh swarm has a population of N particles, then on each iteration, it performs N times the number of distance evaluations than the Michigan swarm.

In Table IX, the results of AMPSO are compared to the results of NN and prototype-based algorithms. For this comparison, AMPSO is used as the reference algorithm for significance tests.

TABLE IX
SUCCESS RATE ON VALIDATION DATA, COMPARISON OF AMPSO VERSUS NN AND PROTOTYPE-BASED ALGORITHMS

	AMPSO	Pittsburgh	IBK 1	IBK	LVQ	ENPC
Problem		PSO	(K=1)	(K=3)		
Balance Scale	85.64	62.79 (-)	82.42 (-)	80.26 (-)	85.10 (=)	**87.00** (+)
Bupa	**65.25**	**65.13** (=)	62.90 (-)	63.16 (-)	62.18 (-)	64.15 (=)
Diabetes	**75.05**	**74.54** (=)	70.44 (-)	73.88 (-)	74.26 (=)	73.54 (-)
Glass	**86.94**	74.34 (-)	71.99 (-)	70.58 (-)	62.56 (-)	82.62 (-)
Iris	**96.89**	90.89 (-)	95.4 (-)	95.2 (-)	95.06 (-)	95.07 (-)
Thyroid	**96.28**	88.93 (-)	**97.21** (=)	93.46 (-)	91.03 (-)	94.81 (-)
Wisconsin	**96.51**	94.43 (-)	95.14 (-)	**96.42** (=)	95.88 (-)	**96.02** (=)

TABLE X
SUCCESS RATE ON VALIDATION DATA, COMPARISON OF AMPSO VERSUS OTHER CLASSIFICATION ALGORITHMS

	AMPSO	J48	PART	Naive	SMO	RBFNN	GAssist	Fuzzy Rule
Problem		(Trees)	(Rules)	Bayes	(SVM)		(GA rules)	Extraction
Bal. Scale	85.64	77.82 (-)	83.17 (-)	**90.53** (+)	87.62 (+)	86.25 (=)	81.81 (-)	78.22 (-)
Bupa	**65.25**	**66.01** (=)	**63.08** (=)	55.63 (-)	57.95 (-)	**65.09** (=)	63.21 (-)	58.42 (-)
Diabetes	75.05	74.48 (=)	74.18 (-)	75.69 (=)	**76.63** (+)	74.37 (=)	73.82 (-)	68.47 (-)
Glass	**86.94**	72.86 (-)	73.79 (-)	47.25 (-)	57.10 (-)	65.45 (-)	61.81 (-)	53.33 (-)
Iris	**96.89**	94.73 (-)	94.20 (-)	95.33 (-)	**96.27** (=)	**96.00** (=)	94.13 (-)	94.06 (-)
Thyroid	**96.28**	92.06 (-)	93.92 (-)	**96.75** (=)	89.74 (-)	**96.41** (=)	92.04 (-)	85.14 (-)
Wisconsin	**96.51**	94.70 (-)	95.14 (-)	**95.99** (=)	**96.85** (=)	**96.08** (=)	**96.03** (=)	95.39 (-)

For comparison, we have used our own experimentation because published studies are not always usable to perform a proper comparison. This can be due to differences in the data sets, normalization procedure and/or validation strategy (leave-one-out, N-fold cross validation, etc.). The WEKA [24] tool was used with this purpose as it is widely used, and the included algorithm implementations are well tested.

Regarding the ENPC algorithm [4], it is an Evolutionary Algorithm with specific operators that allow for prototype reproduction, competition, and extinction. For experimentation, we have used the original implementation from the authors.

Results show that AMPSO performs at least as well as the basic NN classifiers (both 1-NN and 3-NN) on these problems. AMPSO is significantly better than these algorithms in five out of seven cases. The algorithm is able to improve the result because of the placement of the prototypes and probably also due to the elimination of the effect of noise in training patterns. However, this latter point should be tested explicitly in further experiments.

Compared to LVQ and ENPC, AMPSO is also shown to be very competitive, as only ENPC gives a better result for one of the problems.

However, Pittsburgh PSO approach is not so competitive, only being able to produce good results in the Bupa and Balance Scale problems.

The results of AMPSO are compared to the results of commonly used classification algorithms in Table X. For this comparison, AMPSO is used as the reference algorithm for significance tests.

The algorithms in Table X are of two classes. The first five are nonevolutionary algorithms that are based in quite different learning paradigms:

1) J48, an implementation of the C.45 tree-based algorithm;
2) PART, a rule-based classifier;
3) SMO is an SVM method;
4) RBFNN, an implementation of RBFNNs, that have successfully been applied to these problems [18].

The other two algorithms obtain classification rules using an underlying evolutionary algorithm. Experiments with these two algorithms were performed using the Keel [25] tool.

1) GAssist-ADI [19] searches for classification rules encoded using adaptive discrete intervals.
2) Fuzzy Rule Learning Algorithm [20], that extracts fuzzy rules also using a GA.

Results suggest that, for these problems, AMPSO outperforms J48 and PART algorithms, and also both of the GA-based algorithms (GAssist and Fuzzy Rule Extraction). However, Naive Bayes, SMO, and RBFNN are much harder to improve. Overall, AMPSO is the best or equal to the best algorithm in five of the problems.

In the Glass problem, AMPSO is significantly better than any other algorithm. Moreover, ENPC improves all the rest significantly. This suggests that, for this problem, evolutionary placement of prototypes is a good solution. To our knowledge, for this problem, AMPSO has the best success rate in literature.

V. CONCLUSION

The purpose of this paper is to study different versions of PSO applied to continuous classification problems. With this goal, we develop three different versions of Nearest Prototype Classifiers. These algorithms are used to locate a small set of prototypes that represent the data sets used for experimentation without losing classification accuracy.

The first version is an application of the standard PSO, that we call Pittsburgh PSO. In this case, we encode of a full set of prototypes in each particle. However, this produces a search space of high dimension that prevents the algorithm achieving good results. As a first alternative, we propose MPSO, in which each particle represents a single prototype, and the solution is a subset of the particles in the swarm, thus reducing the search space in an effort to obtain better performance.

In both algorithms, a maximum number of prototypes has to be specified for each class in the data set; this is a parameter for the algorithm and its value becomes a limit in the complexity of the solution that may be encoded. To reduce this limitation, we propose a version of MPSO, called AMPSO, which adaptively changes both the number of particles in the swarm and the class distribution of these particles. In this algorithm, only the initial population has to be specified, and the total number of particles of each class may increase during the run of the algorithm.

The MPSO algorithms (both MPSO and AMPSO) introduce a local fitness function to guide the particles' movement and dynamic neighborhoods that are calculated on each iteration. These mechanisms ensure particles do not converge to a single point in the search space. Particles are grouped in neighborhoods that depend on their class; each particle competes and cooperates with the closest neighbors to perform classification of the patterns in its proximity.

We have tested the algorithm in seven well-known benchmark problems that use continuous attributes. We have found that the results of AMPSO and MPSO are always equal to or better than the standard NN classifiers. In most cases, the number of prototypes that compose the solutions found by both MPSO algorithms is quite reduced. This proves that an MPSO can be used to produce a small but representative set of prototypes for a data set. The adaptive version (AMPSO) always produces equal or better solutions than MPSO; it is able to increase accuracy by increasing the number of prototypes in the solution in some of the problems. As for the Pittsburgh PSO, its results never improve the results of the Michigan versions and have higher computational costs.

When the results are compared to other classifiers, AMPSO can produce competitive results in all the problems, specially when used in data sets where the 1-NN classifier does not perform very well. Finally, AMPSO outperforms significantly all the algorithms on the Glass Identification data set, where it achieves more that 10% improvement on average, being the best result found in literature up to this moment on this data set.

It is clear that further work could improve the algorithm performance if it makes AMPSO able to adaptively tune important parameters (such as the reproduction and deletion rate) to the problem. Moreover, any technique that may improve NN classifiers' performance could be applied to AMPSO, such as using a different distance measure or using more than one neighbor for classification.

In summary, our proposed MPSO is able to obtain Nearest Prototype classifiers which provide better or equal results than the most used classification algorithms in problems of different characteristics. Compared to the standard Pittsburgh approach, the Michigan versions provide the following advantages.

1) They attain better results than the Pittsburgh PSO approach due to the reduction in the dimensionality of search space. This means they can also reduce the computational cost.
2) This approach provides the possibility to adjust the complexity of the solution in an adaptive manner. When used (in AMPSO), the resulting algorithm may compete with most of the mainly used classification algorithms.
3) It provides an easy way of implementing competitive and cooperative subsets of the population, as opposed to the Pittsburgh approach in which all the particles interact equally with all their neighbors.

As a result, we think that a Michigan approach for PSO is worth generalization and further investigation in other applications.

References

[1] J. Kennedy, R. Eberhart, and Y. Shi, *Swarm Intelligence*. San Francisco, CA: Morgan Kaufmann, 2001.

[2] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining Knowl. Discovery*, vol. 6, no. 2, pp. 153–172, Apr. 2002.

[3] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Mach. Learn.*, vol. 38, no. 3, pp. 257–286, Mar. 2000. [Online]. Available: citeseer.ist.psu.edu/article/wilson00reduction.html

[4] F. Fernández and P. Isasi, "Evolutionary design of nearest prototype classifiers," *J. Heuristics*, vol. 10, no. 4, pp. 431–454, Jul. 2004.

[5] T. Sousa, A. Silva, and A. Neves, "Particle swarm based data mining algorithms for classification tasks," *Parallel Comput.*, vol. 30, no. 5/6, pp. 767–783, May/Jun. 2004.

[6] Z. Wang, X. Sun, and D. Zhang, *Classification Rule Mining Based on Particle Swarm Optimization*, vol. 4062/2006. Berlin, Germany: Springer-Verlag, 2006.

[7] Z. Wang, X. Sun, and D. Zhang, *A PSO-Based Classification Rule Mining Algorithm*, vol. 4682/2007. Berlin, Germany: Springer-Verlag, 2007.

[8] A. A. A. Esmin, "Generating fuzzy rules from examples using the particle swarm optimization algorithm," in *Proc. HIS*, 2007, pp. 340–343.

[9] N. P. Holden and A. A. Freitas, "A hybrid PSO/ACO algorithm for classification," in *Proc. GECCO Conf. Companion Genetic Evol. Comput.*, 2007, pp. 2745–2750.

[10] A. Cervantes, P. Isasi, and I. Galván, "Binary particle swarm optimization in classification," *Neural Netw. World*, vol. 15, no. 3, pp. 229–241, 2005.

[11] I. D. Falco, A. D. Cioppa, and E. Tarantino, *Evaluation of Particle Swarm Optimization Effectiveness in Classification*. Berlin, Germany: Springer-Verlag, 2006.

[12] J. Holland, "Adaptation," in *Progress in Theoretical Biology*. New York: Academic, 1976, pp. 263–293.

[13] S. W. Wilson, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, no. 2, pp. 149–175, 1995.

[14] A. Cervantes, P. Isasi, and I. Galván, "A comparison between the Pittsburgh and Michigan approaches for the binary PSO algorithm," in *Proc. IEEE CEC*, 2005, pp. 290–297.

[15] A. Cervantes, I. Galván, and P. Isasi, "Building nearest prototype classifiers using a Michigan approach PSO," in *Proc. IEEE SIS*, 2007, pp. 135–140.

[16] T. Kohonen, *Self-Organizing Maps*. Berlin, Germany: Springer-Verlag, 1995.

[17] M. J. D. Powell, "Radial basis functions for multivariable interpolation: A review," in *Algorithms for Approximation of Functions and Data*. Oxford, U.K.: Oxford Univ. Press, 1987, pp. 143–167.

[18] D. Yeung, W. Ng, D. Wang, E. Tsang, and X.-Z. Wang, "Localized generalization error model and its application to architecture selection for radial basis function neural network," *IEEE Trans. Neural Netw.*, vol. 18, no. 5, pp. 1294–1305, Sep. 2007.

[19] J. Bacardit and J. M. Garrell i Guiu, "Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based learning classifier system," in *Proc. GECCO*, 2003, pp. 1818–1831.

[20] H. Ishibuchi, T. Nakashima, and T. Murata, "Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 29, no. 5, pp. 601–618, Oct. 1999.

[21] J. Bratton and D. Kennedy, "Defining a standard for particle swarm optimization," in *Proc. IEEE SIS*, Apr. 1–5, 2007, pp. 120–127.

[22] T. M. Blackwell and P. J. Bentley, "Don't push me! collision-avoiding swarms," in *Proc. IEEE CEC*, 2002, pp. 1691–1696.

[23] T. Blackwell and P. J. Bentley, "Dynamic search with charged swarms," in *Proc. GECCO*, 2002, pp. 19–26.

[24] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, CA: Morgan Kaufmann, 2005.

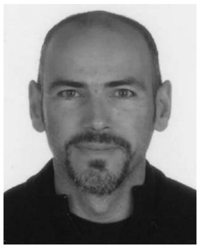
[25] J. Alcala-Fdez, S. Garcia, F. Berlanga, A. Fernandez, L. Sanchez, M. del Jesus, and F. Herrera, "Keel: A data mining software tool integrating genetic fuzzy systems," in *Proc. 3rd Int. Workshop GEFS*, Mar. 2008, pp. 83–88.

Alejandro Cervantes was born in Barcelona, Spain, in 1968. He received the Telecommunications Engineer degree from Universidad Politécnica de Madrid, Madrid, Spain, in 1993. He is currently working toward the Ph.D. degree in the Department of Computer Science, University Carlos III of Madrid.

He is currently an Assistant Teacher with the Department of Computer Science, University Carlos III of Madrid. His current interests focus in swarm intelligence algorithms such as particle swarm, ant colony, and cultural algorithms, both for classification and multiobjective optimization problems.

Inés María Galván received the Ph.D. degree in computer science from Universidad Politécnica de Madrid, Madrid, Spain, in 1998.

From 1992 to 1995, she received a Doctorate-Fellowship, as a Research Scientist, in the European Commission, Joint Research Center Ispra, Italy. Since 1995, she has been with the Department of Computer Science, University Carlos III of Madrid, Madrid, where she has been an Associate Professor since 2000. Her current research focuses in artificial neural networks and evolutionary computation techniques as genetic algorithms, evolutionary strategies, and particle swarm.



Pedro Isasi received the Ph.D. degree in computer science from Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 1990.

He is currently a Professor in computer science with University Carlos III of Madrid, Madrid, Spain, where he is also the Head of the Department of Computer Science and Founder and Director of the Neural Network and Evolutionary Computation Laboratory. His principal research are in the field of machine learning, metaheuristics, and evolutionary optimization methods, mainly applied to the field of finance and economics, forecasting, and classification.