Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 30 (2004) 767–783
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Particle Swarm based Data Mining Algorithms for classification tasks

Tiago Sousa [a,*], Arlindo Silva [a,b], Ana Neves [a,b]

[a] *Escola Superior de Tecnologia, Instituto Politecnico de Castelo Branco, Avenue do Empresario, 6000 Castelo Branco, Portugal*
[b] *Centro de Informatica e Sistemas da Universidade de Coimbra, Polo II—Pinhal de Marrocos, 3030 Coimbra, Portugal*

Received 10 November 2003; accepted 15 December 2003
Available online 18 May 2004

Abstract

Particle Swarm Optimisers are inherently distributed algorithms where the solution for a problem emerges from the interactions between many simple individual agents called particles. This article proposes the use of the Particle Swarm Optimiser as a new tool for Data Mining. In the first phase of our research, three different Particle Swarm Data Mining Algorithms were implemented and tested against a Genetic Algorithm and a Tree Induction Algorithm (J48). From the obtained results, Particle Swarm Optimisers proved to be a suitable candidate for classification tasks. The second phase was dedicated to improving one of the Particle Swarm optimiser variants in terms of attribute type support and temporal complexity. The data sources here used for experimental testing are commonly used and considered as a *de facto* standard for rule discovery algorithms reliability ranking. The results obtained in these domains seem to indicate that Particle Swarm Data Mining Algorithms are competitive, not only with other evolutionary techniques, but also with industry standard algorithms such as the J48 algorithm, and can be successfully applied to more demanding problem domains.

Keywords: Data Mining; Particle Swarm Optimisation; Swarm intelligence

* Corresponding author.
E-mail addresses: tsousa@est.ipcb.pt (T. Sousa), arlindo@est.ipcb.pt (A. Silva), dorian@est.ipcb.pt (A. Neves).

0167-8191/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2003.12.015

1. Introduction

Alongside with the exponential growth of the information technologies, we have witnessed a proliferation of data bases. Nowadays, it is fairly easy to create and customise a data base tailored to our needs. Nevertheless, as the record number grows, it is not that easy to analyse and retrieve high level knowledge from the same data bases. There are not as many off-the-shelf solutions for data analysis as there are for database creation and management, furthermore, they are pretty harder to suit to our needs.

Data Mining (DM) is the most commonly used name to describe such computational analysis and the obtained results must conform to three main requisites: accuracy, comprehensibility and interest for the user [1].

DM comprehends the actions of (semi) automatically seeking out, identifying, validating and using for prediction, structural patterns in data [2], that might be grouped into five categories: decision trees, classification rules, association rules, clusters and numeric prediction.

These patterns are ideally searched for in massive data sets, which could have origins as diverse as genetics, astronomy or agriculture.

Many approaches, methods and goals have been tried out for DM. Biology inspired algorithms such as Genetic Algorithms (GA) and swarm-based approaches like Ant Colonies [3] have been successfully used. In this paper we propose the use of Particle Swarm Optimisers (PSO) in classification rule discovery.

PSO is a new branch in evolutionary algorithms, which were inspired in group dynamics and its synergy and were originated from computer simulations of the coordinated motion in flocks of birds or schools of fish. As these animals wander through a three-dimensional space, searching for food or evading predators, these algorithms make use of particles moving in an n-dimensional space to search for solutions for an n-variable function optimisation problem. In PSO, individuals are called particles and the population is called a swarm [4].

PSO has proved to be competitive with GA in several tasks, mainly in optimisation areas. In phase I of our research we empirically compared three PSO variants with two other algorithms, which have already proven to be reliable in this area: a standard GA and J48—a Java implementation of C4.5. The PSO variants implemented were Discrete Particle Swarm Optimiser [5] (DPSO), Linear Decreasing Weight Particle Swarm Optimiser [6] (LDWPSO) and Constricted Particle Swarm Optimiser [7] (CPSO).

Having established PSO competitiveness in classification rule discovery in phase I, we then focused—on phase II—in optimising one particular variant in terms of temporal complexity and attribute type support.

In Section 2 the underlying structure and algorithms used in our work are explained in detail. Sections 3 and 4 are dedicated to describing the work done in each phase and presenting the experimental setup and obtained results from phases I and II, respectively. Conclusions and future work are in Section 5.

2. Design structure and algorithms

The overall structure of our work was designed to include three-nested algorithms (see Fig. 1); each one fulfils a specific task and is described in details in the following sections.

The innermost algorithm, which is the classification rule discovery algorithm, has for its task to find and return the rule, which better classifies the predominant class in a given instance, set. It is here that the PSO or GA algorithms are used.

The covering algorithm, receives an instance set (the training set), and invokes the classification rule discovery algorithm to reduce this set by removing instances correctly classified by the rule returned by the classification rule discovery algorithm. This process is repeated until a pre-defined number of instances are left to classify in the training set. A sequential rule set is therefore created.

The aim of the validation algorithm—the out most algorithm—is not only to determine the accuracy of a rule set returned by the covering algorithm but also to gauge the liability of the whole classifying algorithm—classification rule discovery and covering algorithms altogether. This is achieved by iteratively dividing the initial data set into different test and training sets and computing average indicators, such as accuracy, time spent, rule number per set and attribute tests number per rule.

2.1. Classification rule discovery algorithm—Particle Swarm Optimisation

As previously mentioned, the rule discovery process is achieved through a PSO algorithm. PSO is inspired in the intelligent behaviour of beings as part of an experience sharing community as opposed to an isolated individual reactive response to the environment. The Adaptive Culture Model [5], which is PSO's framing theory, states that the process of cultural adaptation is rooted into three principles: evaluate, compare and imitate.

Evaluation is the capacity to qualify environmental stimuli and the *sine qua non* condition to social learning. Evaluation itself is both useless and impossible without the ability to compare; all of our metrics are but a comparison to a well-known unit and a single value becomes pointless without the values of its peers. At last, imitation

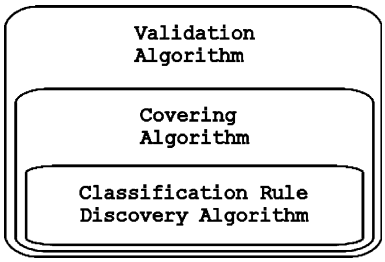

Fig. 1. Three-nested algorithm application structure.

is the rawest form of experience sharing from the receiver's standpoint; it involves not only observation but also the realization of purpose and timing adequacy.

In PSO algorithms, a particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of its most successful neighbour.

There may be different concepts and values for neighbourhood; it can be seen as spatial neighbourhood where it is determined by the Euclidean distance between the positions of two particles, or as a sociometric neighbourhood (e.g.: the index position in the storing array). The latter is the most commonly used for two main motives:

- If space coordinates were to represent mental abilities or skills, two very similar individuals may never come to meet in their lifetime, as to elements of the same family, which may differ significantly from each other, but still, they will always be neighbours.
- The computational effort required to process the Euclidean distance, when faced with large number of particles or dimensions—in each iteration, the distance between every two particles would have to be calculated and for each particle the nearest k neighbours would have to be sorted out.

The number of neighbours (k) usually considered is either $k = 2$ or $k = \text{all}$.

Although some actions differ from one variant of PSO to another, its pseudo-code is as follows:

```
Initiate_Swarm()
  Loop
    For p=1 to number of particles
     Evaluate(p)
     Update_past_experience(p)
     Update_neighbourhood_best(p,k)
     For d=1 to number of Dimensions
       Move(p,d)
Until Criterion.
```

The output of this algorithm is the best point in the hyperspace the swarm visited—and in this case, converged to. There are several variants of PSO, typically differing in the representation: Discrete or Continuous PSO [5]; in the mechanism used to avoid spatial explosion of the swarm and guaranteeing convergence: Linear Decreasing Weight [6] or Constricted PSO [7]; or in the mechanism used to avoid premature convergence to local optima: Predator Prey [8] or Collision Avoiding Swarms [9]. The variants used in our work was the Discrete PSO (DPSO), Constricted PSO (CPSO), Linear Decreasing Weight PSO (LDWPSO) (Fig. 2).

There is a need to maintain and update the particle's previous best position (P_{id}) and the best position in the neighbourhood (P_{gd}). There is also a velocity (V_{id}) associated with each dimension, which is an increment to be made, in each iteration, to

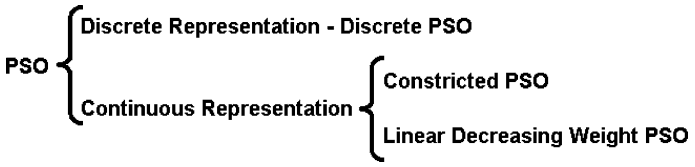

Fig. 2. Implemented PSO algorithms classified accordingly to representation and premature convergence avoiding mechanism.

the dimension associated (Eq. (1)), thus making the particle change its position in the search space.

$$\begin{cases} v_{id}(t) = \chi(v_{id}(t-1) + \varphi_{1id}(P_{id} - x_{id}(t-1)) + \varphi_{2id}(P_{gd} - x_{id}(t-1))) \\ x_{id}(t) = x_{id}(t) + v_{id}(t) \end{cases} \quad (1)$$

φ_1 and φ_2 are random weights defined by an upper limit, χ is a constriction coefficient [7] set to 0.73. The general effect of Eq. (1) is that each particle oscillates in the search space between its previous best position and the best position of its best neighbour, hopefully finding new best points during its trajectory.

If the particle's velocity were allowed to change without bounds the swarm would never converge to an optimum, since particles oscillations would grow larger. The changes in velocity are therefore limited by χ—the constriction coefficient—forcing the swarm to converge.

The value for this parameter and for the upper limits on φ_1 and φ_2 can be chosen to guarantee convergence [7]. In our experiments χ was set to 0.73 while φ_1 and φ_2 upper limits were set to 2.05.

2.2. Classification rule discovery algorithm—Genetic Algorithm

The algorithm used is a standard GA [10], and, as it was used mostly for benchmarking reasons between evolutionary approaches, it will not be explained in detail.

In this algorithm a population of individuals is maintained and evolved according to the principles of natural selection—survival of the fittest. The solution for the problem—here a rule—is encoded in the individual's chromosome, which, in our implementation is a binary string.

Population is evolved by first selecting individuals for mating, using some scheme to guarantee that fitter individuals have a greater probability of generating offspring. As a selection scheme we use tournament selection. New individuals are then generated from the individuals selected for reproduction with the use of two operators: one point crossover, which takes two individuals and returns two new ones which result from exchanging segments of the parents' chromosomes; and mutation, which has a very low probability of flipping any bit in the new individual.

In order to avoid destroying good solutions, a technique called elitism is used: a pre-defined number of the fittest individuals are automatically inserted in the next generation.

2.3. Rule representation

Classification rules are no more than conditional clauses, involving two parts: the antecedent and the consequent. The former is a conjunction of logical tests, and the latter gives the class that applies to instances covered by this rule. These rules take the following format:

```
IF attribute_a = value_1
  AND attribute_b = value_2
  ...
  AND attribute_n = value_i
THEN class_x
```

In rule classifier systems there are two distinct approaches to individual or particle representation: the Michigan and the Pittsburgh approaches [11]. In the Michigan approach each individual encodes a single rule, whereas in the Pittsburgh approach each individual encodes a set of rules. In our work, we follow the Michigan approach.

2.4. Rule evaluation—establishing reference points

Rules must be evaluated during the training process in order to establish points of reference for the training algorithm: best particle positioning. The rule evaluation function must not only consider instances correctly classified but also the ones left to classify and the wrongly classified ones.

The formula used to evaluate a rule and therefore set its quality is expressed in Eq. (2) [12]:

$$Q(X) = \begin{cases} \frac{\mathrm{TP}}{\mathrm{TP+FN}} \times \frac{\mathrm{TN}}{\mathrm{TN+FP}} & \text{if } 0.0 \leqslant x_i \leqslant 1.0, \forall i \in d \\ -1.0 & \text{otherwise,} \end{cases} \tag{2}$$

where:

- TP—True Positives = number of instances covered by the rule that are correctly classified, i.e., its class matches the training target class.
- FP—False Positives = number of instances covered by the rule that are wrongly classified, i.e., its class differs from the training target class.
- TN—True Negatives = number of instances not covered by the rule, whose class differs from the training target class.
- FN—False Negatives = number of instances not covered by the rule, whose class matches the training target class.

This formula penalizes a particle, which as moved out of legal values, assigning it with negative value (−1.0), forcing it to return to the search space.

2.5. Covering algorithm—rule set construction

The covering algorithm is basically a divide-and-conquer technique. Being given a instance training set, it runs the rule discovery algorithm in order to obtain the highest quality rule for the predominant class in the training set.

Once found, this rule goes through a pruning process where unnecessary attribute tests are removed. This is a simple process that iteratively removes attribute tests if the quality of the obtained rule has the same or an higher value than the original rule.

Correctly classified instances are then removed from the training set and the rule discovery algorithm is run once more. Iteratively a sequential rule set is built, and the covering algorithm runs until only a pre-defined number of instances are left to classify. This threshold criteria value is user-defined as a percentage and it is typically set to 10%.

A default rule, to capture and classify instances not classified by the previous rules is added to the rule set. Containing no attribute tests and predicting the same class as the one predominant in the remaining instances, this rule takes the form:

```
IF true
THEN class_x.
```

2.6. Validation algorithm— rule set and overall evaluation

The purpose of the validation algorithm is to statistically evaluate the accuracy of the rule set obtained by the covering algorithm. This is done using a method known as tenfold cross-validation [2].

The tenfold cross validation consists in dividing the data set into 10 equal partitions and iteratively using one of this sets as a test set and the remaining nine as training sets. In the end 10 different rule sets are obtained and average indicators, such as accuracy, time spent, rule number per set and attribute tests number per rule are computed.

Several other numbers for partitioning have been tried out, but theoretical research [2] has shown that 10 offers the best estimate of errors.

Rule set accuracy is evaluated and presented as the percentage of instances in the test set correctly classified. An instance is considered correctly classified, when the first rule in the rule set, whose antecedent matches this instance and the consequent (predicted class) matches this instance's class.

2.7. Pre-processing routines—data extraction

In a pre-processing routine, the original data set is extracted from file, parsed and analyzed. Two data structures are created: a normalized image of the data set and a structure containing metadata information.

A state attribute is assigned to each instance. Manipulating this state value, it is very easy and computationally efficient, to divide the data set into training and test sets and to (pseudo-) remove instances. This attribute takes the following values: TEST, TRAIN and REMOVED.

2.8. Post-processing routines—rule pruning and rule set cleaning

Recall that high level knowledge extracted from databases must conform to three main requisites: accuracy, comprehensibility and interested for the user [1].

In classification rule discovery problems, the number of attribute tests per rule and the number of rules per set is a major contributor for the comprehensibility of the obtained results—fewer attribute tests and rules eases comprehensibility.

After a rule is returned from the classification rule discovery algorithm it goes through a pruning process in order to remove unnecessary attribute tests. This is done by iteratively removing each attribute test whenever the newly obtained rule has the same or higher quality value than the original rule.

Just after the covering algorithm returns a rule set, another post-processing routine is used: rule set cleaning, where rules that will never be applied are removed from the rule set.

As rules in the rule set are applied sequentially, in this routine, rules are removed from the rule set if:

- There is a previous rule in the rule set that has a subset of the rule's attribute tests.
- If it predicts the same class as the default rule and is located just before it.

So in the example below, rules number 2 and 3 will be removed and the rule set will be reduced to the first and last rules:

```
Rule #1
  If attribute_a = x_a
  Then class = c_1
Rule #2
  If attribute_a = x_a and attribute_b = x_b
  Then class = c_2
Rule #3
  If attribute_c = x_c
  Then class = c_3
Rule #4 - Default Rule
  If TRUE
  Then class = c_3.
```

3. Phase I—testing Particle Swarm Optimisation in Data Mining

3.1. Pre-processing routines—data normalization

Only nominal attributes were supported in this phase and were normalized assigning to each different attribute value an enumerated index. Lookup tables were kept as metadata information. When parsing an instance, only this index is stored in the normalised data image.

3.2. Rule representation

Binary string representation is a requisite for the DPSO, so in this phase rules were coded in binary strings in order to provide a fair experimental testing platform for all implemented algorithms.

In our initial approach indifference was implemented with an extra attribute value and applying Eq. (3) to determine the binary string length. Attribute test matching would occur accordingly to Eq. (4):

$$Rule\_bit\_number = \sum_{a=1}^{number\_of\_attributes} \lceil \log_2(1 + different\_values_a) \rceil \tag{3}$$

$$m(v_{ra}, v_{ia}) = \begin{cases} \text{true} & \text{if } v_{ra} = v_{ia} \text{ or } v_{ra} \geqslant different\_values_a \\ \text{false} & \text{otherwise.} \end{cases} \tag{4}$$

Being v_{ra} the attribute value stored in the rule's binary sub-string for attribute a and v_{ia} the instance indexed value stored in the normalized image of the data set.

Nevertheless, this approach revealed to be unbalanced, regarding indifference probability occurrence, a more even approach was obtained assigning an extra bit for indifference (Eq. (5)) as it can be seen in Fig. 3.

$$Rule\_bit\_number = \sum_{a=1}^{number\_of\_attributes} \lceil 1 + \log_2 different\_values_a \rceil \tag{5}$$

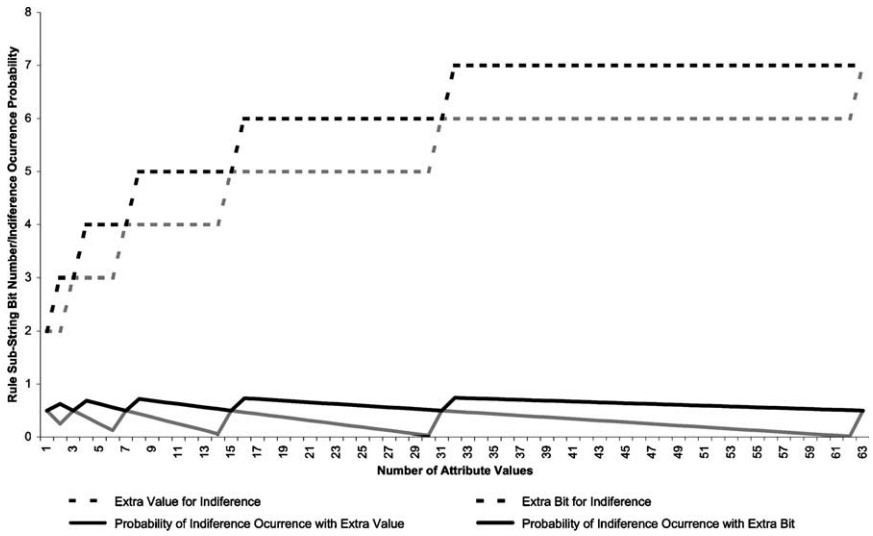

Fig. 3. Indifference occurrence probability with one value and one extra bit.

3.3. Classification rule discovery algorithm—loop end criteria

Inevitably, with more or less iterations, the swarm converges to an optimum (possibly just a local one) (see Fig. 4).

In this phase the criteria used to trigger the ending of the loop was the realization that such convergence was achieved. This was done by monitoring the best particle's quality and stopping when it had been constant for a safe number of iterations.

In our work the number of iterations needed to trigger the ending of the loop has been named as **Convergence Platform Width** and for convenience we refer to this criteria in the same way.

3.4. Experimental setup

For each experiment the complete instance set was divided into 10 equal parts; iteratively each of these parts is used as a test set and the remaining nine as training sets. This is done in order to provide a significant averaged measure of the algorithm performance. This procedure is called tenfold cross-validation; the number of folds has been subjected to testing [2], and tenfold cross-validation is currently considered a standard evaluation procedure in DM.

Different numbers of particles/individuals were tested since previous research has suggested that PSO needs fewer particles than GA does need individuals to obtain the same results, thus demanding fewer resources.

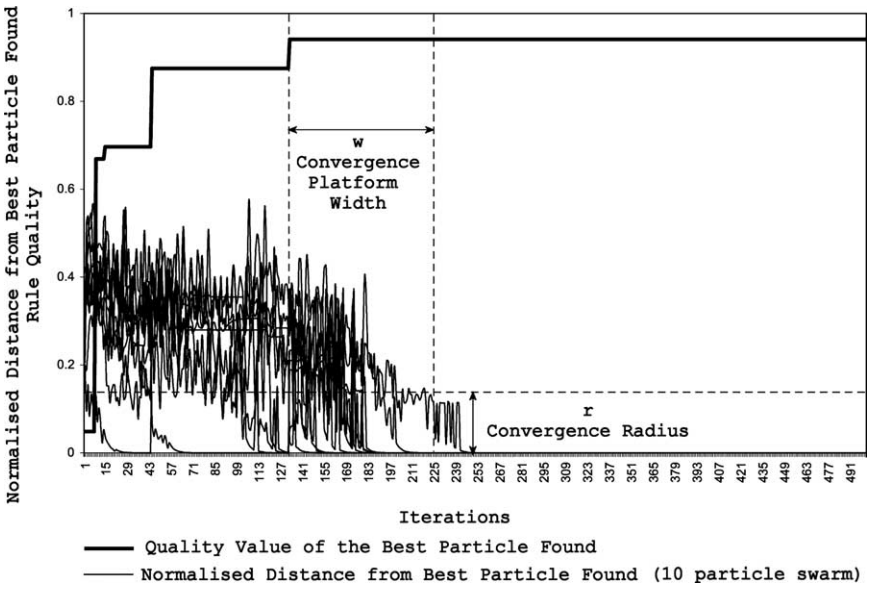

Fig. 4. Loop end criteria.

Convergence Platform Width was set to 30 and maximum uncovered instances to 10%.

Trees obtained with J48 are easily converted to rules—each path from the root to a leaf stands for a rule, each node stands for one attribute test and the leaf is the rule's consequent or predicted class.

The data sources used were obtained from the Department of Computer Science, University of Waikato, Hamilton, New Zealand [13], and Information and Computer Science, University of California [14].

3.5. Results and discussion

In Tables 1–3 we present the experimental results obtained in phase I. Accuracy values are in percentage of success, and are obtained by averaging tenfold accuracy results, with standard deviation. An average of accuracy from all the different population numbers is also presented to ease analysis.

Three data sets were used: Zoo, Breast-Cancer and Wisconsin-Breast-Cancer. Zoo is a data set that classifies animals according to their characteristics. Breast-Cancer and Wisconsin-Breast-Cancer are real data sets that classify if the tumour was malignant/benign and if recurrence of events did happen.

Table 1
Phase I: experimental results for relation Zoo

Population	DPSO	CPSO	LDWPSO	GA	J48
	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy
25	88 ± 6	81 ± 13	70 ± 14	89 ± 7	92
50	91 ± 5	80 ± 8	84 ± 12	91 ± 7	
100	92 ± 6	85 ± 10	86 ± 8	89 ± 7	
200	94 ± 6	85 ± 9	87 ± 10	87 ± 9	
300	90 ± 6	93 ± 7	91 ± 5	91 ± 7	
	91	85	84	89	

Table 2
Phase I: experimental results for relation Breast-Cancer

Population	DPSO	CPSO	LDWPSO	GA	J48
	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy
25	73 ± 7	74 ± 6	77 ± 7	74 ± 6	73
50	75 ± 7	72 ± 7	74 ± 6	75 ± 7	
100	77 ± 5	76 ± 7	75 ± 8	75 ± 7	
200	77 ± 5	74 ± 6	76 ± 6	77 ± 6	
300	77 ± 5	75 ± 7	75 ± 7	76 ± 6	
	76	74	75	75	

Table 3
Phase I: experimental results for relation Wisconsin-Breast-Cancer

Population	DPSO	CPSO	LDWPSO	GA	J48
	Accuracy	Accuracy	Accuracy	Accuracy	Accuracy
25	94 ± 3	93 ± 3	91 ± 5	94 ± 4	93
50	94 ± 3	93 ± 4	92 ± 6	93 ± 3	
100	94 ± 3	94 ± 3	94 ± 2	93 ± 4	
200	93 ± 5	93 ± 4	94 ± 4	94 ± 3	
300	93 ± 3	94 ± 3	94 ± 4	93 ± 3	
	94	93	93	93	

Results obtained clearly state the competitiveness of PSO with industrial tree induction algorithms like J48, a Java implementation of C45.

In the Zoo data set (Table 1), average results are inferior to J48, nevertheless in Breast-Cancer data set (Table 2) and Wisconsin-Breast-Cancer data set (Table 3), average results are equal or slightly superior to J48 leading to conclude that regarding accuracy PSO can compete both with other evolutionary approaches and more classical techniques.

From these results, we could conclude that an increase of particle/individual above 25 does not bring any relevant improvement in the algorithm's performance. All evolutionary algorithms are equally efficient with a low number of particles/individuals, i.e. they have low spatial complexity.

We could also conclude that PSO based DM can compete both with other evolutionary approaches and more classical techniques, at least in some data sets, in terms, not only of accuracy, but also of spatial complexity. Nevertheless, all implemented algorithms performed very poorly, regarding the time spent (results are presented in phase II), creating therefore the need for temporal complexity optimisation.

4. Phase II—optimising the classification rule discovery algorithm

Support for attribute types other than nominal urged in order to move to more demanding areas and data sets.

We opted for the CPSO variant (see Fig. 2). Due to its continuous representation, as opposing the binary representation of the DPSO variant, the Constricted variant proved to be more qualified when it came to dealing with numeric attributes (both integer and real). Although the LDWPSO does also have a continuous representation, it needs a fixed number of iterations, leaving little room for temporal complexity optimisation, which was another of our concerns, for without optimisation in this area, expansion to more demanding problems is seriously affected or even made impossible.

Following the experimental platform, of the previous phase, the same data sets were used, in order to assert whether this approach could offer significant improvements.

4.1. Pre-processing routines—data extraction and normalization

As mentioned before support for numeric—integer and real—attributes was added.

All attribute values are normalized to the range $[0.0, t]$ with $0.0 < t < 1.0$, being t a user pre-defined value, it stands for the indifference threshold where a higher value will trigger the omission of the corresponding attribute test.

This normalised value is used not only in the data image but also in space dimensions, avoiding costly conversions to detect attribute matching. The use of this normalised value throughout data image, rule/dimensions, particle moving and attribute matching was our strongest bet to temporal complexity optimisation.

Nominal attributes are normalised assigning to each different attribute value an enumerated index $\#idx$ and applying Eq. (6).

$$v_{\text{norm}} = \frac{idx_v \times t}{\#idx}. \tag{6}$$

idx_v is the index of the attribute value v and $\#idx$ the total number of different attribute values. Both integer and real types are normalized with Eq. (7).

$$v_{\text{norm}} = \frac{(v - v_{\min}) \times t}{v_{\max} - v_{\min}}. \tag{7}$$

$v_{\min}$ and $v_{\max}$ are the lower and higher attribute values found for this attribute.

4.2. Rule representation

In phase I attribute testing, indifference was implemented with an extra bit and particles were coded in binary strings, as a result indifference probability occurrence would vary accordingly to the attribute assigned bit number and its range of possible values, in the interval]1/2, 3/4] (see Fig. 3).

In this phase a user defined threshold level maintains attribute-testing indifference, therefore different values for this threshold were tested in order to evaluate its influence.

Rules are encoded as a floating-point array and each attribute is represented by either one or two elements on the array, according to its type: nominal attributes are assigned with one element on the array and attribute-matching tests are defined as follows:

$$m(v_{\text{r}}, v_{\text{i}}) = \begin{cases} \text{true} & \text{if } \lfloor v_{\text{r}} \times \#idx \rfloor = \lfloor v_{\text{i}} \times \#idx \rfloor \\ \text{false} & \text{otherwise.} \end{cases} \tag{8}$$

Being t the indifference threshold value, v_{r} the attribute value stored in the rule for testing and v_{i} the instance value stored in the normalized image of the data set.

Integer and real attributes are assigned with an extra element in the array in order to implement a value range instead of a single value

$$m(v_{\text{r1}}, v_{\text{r2}}, v_{\text{i}}) = \begin{cases} \text{true} & \text{if } v_{\text{r1}} \geqslant t \text{ or } (v_{\text{r1}} - v_{\text{r2}}) \leqslant v_{\text{i}} \text{ or } (v_{\text{r1}} + v_{\text{r2}}) \geqslant v_{\text{i}} \\ \text{false} & \text{otherwise.} \end{cases} \tag{9}$$

v_{r1} can be seen as the center and v_{r1} as a neighbourhood radius, inside which matching will occur.

4.3. Classification rule discovery algorithm—loop end criteria

In phase I the criteria used to trigger the ending of the PSO's main loop was the Convergence Platform Width, is this phase it is the realization that all particles in the swarm are within a user-defined distance from the best particle in the swarm. As this distance is no more than the radius of an hypersphere this criteria is here referred to as **Convergence Radius**.

In order to manipulate equivalent threshold distances, considering that distance ranges will differ accordingly to the dimension number, the distance formula used is the normalized Euclidean distance

$$d(p_1, p_2) = \frac{\sqrt{\sum_{i=1}^{n} (p_1^i - p_2^i)^2}}{\sqrt{d}.} \tag{10}$$

p_1 and p_2 are particles and d the dimension number, p_n^i stands for the ith coordinate value of particle p_n. As each dimension coordinate is bounded to the interval [0.0, 1.0] the maximum value for $(p_1^i - p_2^i)$ is 1.0, which when squared remains 1.0, therefore to normalize a distance, all that is needed is to divide it by $\sqrt{d}$.

4.4. Experimental setup

We aimed to test phase I against phase II, both loop-end criteria and evaluate the influence of different indifference threshold levels.

To maintain a fair experimental platform with phase I the same data sources were used: Zoo, Breast-Cancer and Winsconsin-Breast-Cancer. Also tenfold cross-validation and the same number of runs were used.

The swarms were set to 25 particles, Convergence Radius to 0.1. As used in phase I Convergence Platform Width was set to 30 and maximum uncovered instances to 10%.

Indifference threshold was tested with 3 values, 0.1, 0.5 and 0.7, in order to assert its possible influence.

4.5. Results and discussion

In Tables 4–6 we present the experimental results obtained in this phase. Accuracy values are in percentage of success/accuracy, and are obtained by averaging tenfold accuracy results. In Table 7 we compare results obtained in each phase regarding the time spent, presenting the time spent by each CPSO version and its reason (phase I/phase II).

Regarding temporal complexity optimisation, the new CPSO version clearly outperformed the one implemented in phase I (see Table 7) with up to 26 times faster.

Table 4
Relation Zoo

		Phase I	Phase II					
	J48	CPSO	Platform width			Radius		
Indifference	–	–	0.1	0.5	0.7	0.1	0.5	0.7
Accuracy	92.0	84.8	89.3	66.7	45.7	89.0	77.0	46.7
Time spent	0.09	11.75	0.45	0.87	0.63	17.36	13.01	3.35
Number of rules	13	7	7	6	5	6	6	5
Tests per rule	5	2	5	6	4	5	4	4

Table 5
Relation Breast-Cancer

		Phase I	Phase II					
	J48	CPSO	Platform width			Radius		
Indifference	–	–	0.1	0.5	0.7	0.1	0.5	0.7
Accuracy	72.9	74.2	74.3	74.8	72.6	76.7	75.1	73.3
Time spent	0.03	7.04	0.57	1.31	1.73	28.13	13.19	17.81
Number of rules	4	5	5	5	5	6	6	6
Tests per rule	2	2	4	4	3	5	5	5

Table 6
Relation Winsconsin-Breast-Cancer

		Phase I	Phase II					
	J48	CPSO	Platform width			Radius		
Indifference	–	–	0.1	0.5	0.7	0.1	0.5	0.7
Accuracy	92.9	93.4	92.9	88.7	87.7	92.8	91.8	76.6
Time spent	0.02	17.34	1.53	4.80	36.82	85.01	67.16	18.05
Number of rules	55	7	7	9	104	7	7	5
Tests per rule	2	1	4	7	8	4	4	4

Table 7
Time spent—comparing phase I against phase II (convergence platform width)

	Zoo				Breast-Cancer				Winsconsin-Breast-Cancer			
	I	II			I	II			I	II		
Indifference	–	0.1	0.5	0.7	–	0.1	0.5	0.7	–	0.1	0.5	0.7
Time	11.75	0.45	0.87	0.63	7.04	0.57	1.31	1.73	17.34	1.53	4.80	36.82
Reason	–	26.11	13.50	18.65	–	12.35	5.37	4.07	–	11.34	3.61	0.47

There is a clear relation between indifference threshold level value and accuracy results: best results were obtained with lower values for indifference threshold level.

Regarding accuracy, both CPSO versions did surpass J48 in the Wisconsin-Breast-Cancer relation. Nevertheless, temporal complexity of both CPSO versions

are still much more demanding than J48, possibly due to the nature of the algorithm and processing involved.

Comprehensibility of results is related with fewer rules and attribute tests. Very good results were obtained with both versions. Here too we can establish a relation with the indifference threshold level: best results were obtained with lower values for indifference threshold level.

5. Conclusions and future work

In phase I we proposed to test PSO in DM tasks, namely classification rule discovery, and empirically compared the results with another evolutionary algorithm (GA) and with J48, a Java implementation of C4.5.

Based on the obtained results, phase II was dedicated to improve one of the PSO variants investigated. Our goals in this phase were temporal complexity optimisation, attribute type support expansion and evaluation of the possible influences of indifference threshold values. We implemented and compared this variant with the corresponding one in phase I and J48, in some benchmark data.

From the results, we can conclude that PSO can obtain competitive results against J48 in the data sets used, although there is some increase in the computational effort needed. We can also conclude, that lower values for indifference threshold offer the best accuracy results.

Directions for future work include an empirical analysis of the influence of indifference threshold with exploration and exploitation and applying this tool to more demanding data sources—containing continuous attributes.

References

[1] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery: an overview, in: Advances in Knowledge Discovery and Data Mining, AAAI/MIT, Cambridge, 1996, pp. 1–34.
[2] Ian H. Witten, E. Frank, Data Mining—Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kauffmann, 1999.
[3] R. Parpinelli, H. Lopes, A. Freitas, An Ant Colony Algorithm for Classification Rule Discovery, Idea Group, 2002.
[4] J. Kennedy, R.C. Eberhart, Particle Swarm Optimisation, in: Proceedings of the IEEE, International Conference on Neural Networks, Piscataway, 1995.
[5] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kauffman, 2001.
[6] Y. Shi, R.C. Eberhart, Empirical Study of Particle Swarm Optimisation, in: Proceedings of the 1999 Congress of Evolutionary Computation, Piscatay, 1999.
[7] M. Clerc, J. Kennedy, The particle swarm-explosion, stability and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002).
[8] A. Silva, A. Neves, E. Costa, Chasing the Swarm: A Predator Prey Approach to Function Optimisation, in: Proceedings of the MENDEL2002—8th International Conference on Soft Computing. Brno, Czech Republic, 2002.
[9] T. Blackwell, P.J. Bentley, Don't Push Me! Collision-Avoiding Swarms. in: Proceedings of the Congress on Evolutionary Computation, 2002.

[10] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, Massachusetts, 1989.
[11] A. Freitas, A survey of evolutionary algorithms for data mining and knowledge discovery, in: A. Ghosh, S. Tsutsui (Eds.), Advances in Evolutionary Computation, Springer-Verlag, 2001.
[12] H.S. Lopes, M.S. Coutinho, W.C.: An Evolutionary Approach to Simulate Cognitive Feedback Learning in Medical Domain. Genetic Algorithms and Fuzzy Logic Systems: Soft Computing Perspectives. ISBN 981-02-2423-0, World Scientific, Singapore 1997.
[13] ftp://ftp.cs.waikato.ac.nz/pub/ml/datasets-UCI.jar.
[14] http://www.ics.uci.edu/cmerz/mldb.tar.Z.