Guilherme Bittencourt
Geber L. Ramalho (Eds.)

LNAI 2507

Advances in Artificial Intelligence

16th Brazilian Symposium on Artificial Intelligence, SBIA 2002
Porto de Galinhas/Recife, Brazil, November 2002
Proceedings

Springer

Mining Comprehensible Rules from Data with an Ant Colony Algorithm

Rafael S. Parpinelli[1], Heitor S. Lopes[1], and Alex A. Freitas[2]

[1] CEFET-PR, CPGEI
Av. Sete de Setembro, 3165, Curitiba - PR, 80230-901, Brazil
{rsparpin,hslopes}@cpgei.cefetpr.br
[2] PUC-PR, PPGIA-CCET
R. Imaculada Conceição, 1155, Curitiba - PR, 80215-901, Brazil
alex@ppgia.pucpr.br

Abstract. This work describes an algorithm for data mining called Ant-Miner (Ant Colony-based Data Miner). The goal of Ant-Miner is to extract classification rules from data. The algorithm is inspired by both research on the behavior of real ant colonies and some data mining concepts and principles. We compare the performance of Ant-Miner with CN2, a well-known data mining algorithm for classification, in six public domain data sets. The results provide evidence that: (a) Ant-Miner is competitive with CN2 with respect to predictive accuracy; and (b) The rule lists discovered by Ant-Miner are considerably simpler (smaller) than those discovered by CN2.

1 Introduction

In essence, the goal of data mining is to extract knowledge from data. We emphasize that in data mining – unlike for example in classical statistics – the goal is to discover knowledge that is not only accurate but also comprehensible for the user [8], [9], [10]. Comprehensibility is important whenever discovered knowledge will be used for supporting a decision made by a human user. After all, if discovered knowledge is not comprehensible for the user, he/she will not be able to interpret and validate it. In this case, probably the user will not trust enough the discovered knowledge to use it for decision making. This can lead to wrong decisions.

In this paper we describe an Ant Colony-based data mining algorithm for the classification task of data mining. In this task the goal is to assign each case (object, record, or instance) to one class, out of a set of predefined classes, based on the values of some attributes (called predictor attributes) for the case. For the classification task, the discovered knowledge is often represented in the form of *IF* <*conditions*> *THEN* <*class*> rules (which are further discussed in Section 3).

To the best of our knowledge the use of Ant Colony Optimization (ACO) algorithms [5] for discovering classification rules, in the context of data mining, is a research area still unexplored. Actually, the only Ant Colony-based algorithm

G. Bittencourt and G. Ramalho (Eds.): SBIA 2002, LNAI 2507, pp. 259-269, 2002.

developed for data mining that we are aware of is an algorithm for clustering [13], which is, of course, a data mining task very different from the classification task addressed in this paper.

We believe the development of ACO algorithms for data mining is a promising research area, due to the following reason. ACO algorithms involve simple agents (ants) that cooperate with one another to achieve an emergent, unified behavior for the system as a whole, producing a robust system capable of finding high-quality solutions for problems with a large search space. In the context of rule discovery, an ACO algorithm has the ability to perform a flexible, robust search for a good combination of terms (logical conditions) involving values of the predictor attributes.

2 Ant Colony Optimization

An Ant Colony Optimization algorithm (ACO) is essentially a system based on agents which simulate the natural behavior of ants, including mechanisms of cooperation and adaptation. This new heuristic was proposed in [6] in order to solve combinatorial optimization problems and it has been shown to be both robust and versatile – in the sense that it has been successfully applied to a range of different combinatorial optimization problems [7]. In passing we mention that recently there has been a growing interest in developing rule-discovery algorithms based on other kinds of bio-inspired algorithms – mainly evolutionary algorithms [10]. ACO algorithms are based on the following ideas:

- Each path followed by an ant is associated with a candidate solution for a given problem;
- When an ant follows a path, the amount of pheromone (a chemical substance used in real ant colonies) deposited on that path is proportional to the quality of the corresponding candidate solution for the target problem;
- When an ant has to choose between two or more paths, the path(s) with a larger amount of pheromone (i.e., the path(s) that were more frequently chosen by other ants in the past) have a greater probability of being chosen by the ant.

As a result, the ants eventually converge to a short path, hopefully the optimum or a near-optimum solution for the target problem.

In essence, the design of an ACO algorithm involves the specification of [1]:

- An appropriate representation of the problem, which allows the ants to incrementally construct/modify solutions through the use of a probabilistic transition rule, based on the amount of pheromone in the trail and on a local, problem-dependent heuristic;
- A method to enforce the construction of valid solutions, that is, solutions that are legal in the real-world situation corresponding to the problem definition;
- A problem-dependent heuristic function (η) that measures the quality of items that can be added to the current partial solution;
- A rule for pheromone updating, which specifies how to modify the pheromone trail (τ);

- A probabilistic transition rule based on the value of the heuristic function (η) and on the pheromone trail (τ) that is used to iteratively construct a solution.

3 Ant-Miner: A New ACO Algorithm for Data Mining

In an ACO algorithm each ant incrementally constructs/modifies a solution for the target problem. In our case the target problem is to discover classification rules. As mentioned in the introduction, each classification rule has the form: *IF* < *term1* AND *term2* AND ...> *THEN* <*class*>. Each term is a triple <*attribute, operator, value*>, where *value* is a value belonging to the domain of *attribute*. The operator element in the triple is a relational operator. The current version of Ant-Miner copes only with categorical attributes, so that the operator element in the triple is always "=". Continuous (real-valued) attributes are discretized in a preprocessing step.

A high-level description of Ant-Miner is shown in Algorithm I.

TrainingSet = {all training cases};
DiscoveredRuleList = []; /* rule list is initialized with an empty list */
WHILE (TrainingSet > Max_Uncovered_Cases)
  i = 1; /* ant index */
  j = 1; /* convergence test index */
  Initialize all trails with the same amount of pheromone;
REPEAT
Ant_i starts with an empty rule and incrementally constructs a classification rule R_i, by adding one term at a time to the current rule;
Prune rule R_i;
Update the pheromone of all trails, by increasing pheromone in the trail followed by Ant_i (proportional to the quality of R_i) and decreasing pheromone in the other trails (simulating pheromone evaporation);
IF (R_i is equal to R_{i-1}) /* update convergence test */
THEN j = j + 1;
ELSE j = 1;
END IF
i = i + 1;
UNTIL (i ≥ No_of_Ants) OR (j ≥ No_Rules_Converg)
Choose the best rule R_{best} among all rules R_i constructed by all the ants;
Add rule R_{best} to DiscoveredRuleList;
TrainingSet = TrainingSet - {set of cases correctly covered by R_{best}};
END WHILE

Algorithm 1. A High-Level Description of Ant-Miner

Ant-Miner follows a sequential covering approach to discover a list of classification rules covering all or almost all training cases. At first, the list of discovered rules is empty and the training set consists of all training cases. Each iteration of the WHILE loop of Algorithm I, corresponding to a number of executions of the REPEAT-UNTIL loop, discovers one classification rule. This rule is added to

the list of discovered rules, and the training cases that are correctly covered by this rule (i.e., cases satisfying the rule antecedent and having the class predicted by the rule consequent) are removed from the training set. This process is iteratively performed while the number of uncovered training cases is greater than a user-specified threshold, called *Max_Uncovered_Cases*. Each iteration of the REPEAT-UNTIL loop of Algorithm I consists of three steps, comprising rule construction, rule pruning, and pheromone updating, discussed in the following.

3.1 Rule Construction

For the rule construction step, the Ant_i starts with an empty rule, that is, a rule with no term in its antecedent, and adds one term at a time to its current partial rule. The current partial rule constructed by an ant corresponds to the current partial path followed by that ant. Similarly, the choice of a term to be added to the current partial rule corresponds to the choice of the direction in which the current path will be extended. The choice of the term to be added to the current partial rule depends on both a problem-dependent heuristic function (η) and on the amount of pheromone (τ) associated with each term. Ant_i keeps adding one-term-at-a-time to its current partial rule until one of the following two stopping criteria is met:

- Any term to be added to the rule would make the rule cover a number of cases smaller than a user-specified threshold, called *Min_cases_per_rule* (minimum number of cases covered per rule). This enforces at least a certain degree of generality in the discovered rules, helping to avoid an overfitting to the training data; or
- All attributes have already been used by the ant, so that there is no more attributes to be added to the rule antecedent. Notice that each attribute can occur only once in each rule, to avoid invalid rules such as "IF (Sex = male) AND (Sex = female)....".

This process is repeated until one of the two following conditions is met:

- The number of constructed rules is equal to or greater than the user-specified threshold *No_of_Ants*;
- The current Ant_i has constructed a rule that is exactly the same as the rule constructed by the previous *No_Rules_Converg* – 1 ants, where *No_Rules_Converg* stands for the number of rules used to test convergence of the ants, this is, whether the ants converged to a single rule (path).

Let $term_{ij}$ be a rule condition of the form $A_i = V_{ij}$, where A_i is the i-th attribute and V_{ij} is the j-th value of the domain of A_i. The probability that $term_{ij}$ is chosen to be added to the current partial rule is given by Equation (1):

$$P_{ij} = \frac{\eta_{ij} \cdot \tau_{ij}(t)}{\sum_{i=1}^{a} x_i \cdot \sum_{j=1}^{b_i} \left(\eta_{ij} \cdot \tau_{ij}(t)\right)} \quad (1)$$

where:

- η_{ij} is the value of a problem-dependent heuristic function for $term_{ij}$;
- $\tau_{ij}(t)$ is the amount of pheromone associated with $term_{ij}$ at time t, corresponding to the amount of pheromone currently available in the position i,j of the path being followed by the current ant;
- a is the total number of attributes;
- x_i is set to 1 if the attribute A_i was not yet used by the current ant, or to 0 otherwise;
- b_i is the number of values in the domain of the i-th attribute.

Once the rule antecedent is completed, the system chooses the rule consequent (i.e., the predicted class) that maximizes the quality of the rule. This is done by assigning to the rule consequent the majority class among the cases covered by the rule.

3.2 Heuristic Function

For each $term_{ij}$ that can be added to the current rule, Ant-Miner computes the value η_{ij} of a heuristic function that is an estimate of the quality of this term, with respect to its ability to improve the predictive accuracy of the rule. This heuristic function is based on Information Theory [4]. More precisely, the value of η_{ij} for $term_{ij}$ involves a measure of the entropy (or amount of information) associated with that term. For each $term_{ij}$ its entropy is

$$H(W|A_i = V_{ij}) = -\sum_{w=1}^{k}\left(P(w|A_i = V_{ij}) \cdot \log_2 P(w|A_i = V_{ij})\right) \quad (2)$$

where:

- W is the class attribute (i.e., the attribute whose domain consists of the classes to be predicted);
- k is the number of classes;
- $P(w|A_i=V_{ij})$ is the empirical probability of observing class w conditional on having observed $A_i=V_{ij}$.

The higher the value of $H(W|A_i=V_{ij})$, the more uniformly distributed the classes are, and so the smaller the probability of the current ant choosing $term_{ij}$ to be added to its partial rule. It is desirable to normalize the value of the heuristic function to facilitate its use in Equation (1), combining both this function and the amount of pheromone. In order to implement this normalization, it is used the fact that the value of $H(W|A_i=V_{ij})$ varies in the range $0 \le H(W|A_i=V_{ij}) \le \log_2 k$, where k is the number of classes. Therefore, the proposed normalized, information-theoretic heuristic function is:

$$\eta_{ij} = \frac{\log_2 k - H(W|A_i = V_{ij})}{\sum_{i=1}^{a} x_i \cdot \sum_{j=1}^{b_i}\left(\log_2 k - H(W|A_i = V_{ij})\right)} \quad (3)$$

where:

- a is the total number of attributes;
- x_i is set to 1 if the attribute A_i was not yet used by the current ant or to 0 otherwise;
- b_i is the number of values in the domain of the i-th attribute.

Hence, the higher the value of η_{ij}, the more relevant for classification the $term_{ij}$ is, and so the higher its probability of being chosen.

In the above heuristic function there are just two minor caveats. First, if the value V_{ij} of attribute A_i does not occur in the training set then $H(W|A_i=V_{ij})$ is set to its maximum value of $log_2 k$. This corresponds to assigning to $term_{ij}$ the lowest possible predictive power. Second, if all the cases belong to the same class then $H(W|A_i=V_{ij})$ is set to 0. This corresponds to assigning to $term_{ij}$ the highest possible predictive power.

3.3 Rule Pruning

Rule pruning is a commonplace technique in data mining [2]. The main goal of rule pruning is to remove irrelevant terms that might have been unduly included in the rule. Rule pruning potentially increases the predictive power of the rule, helping to avoid its overfitting to the training data. Another motivation for rule pruning is that it improves the simplicity of the rule, since a shorter rule is usually easier to be understood by the user than a longer one.

As soon as the current ant completes the construction of its rule, the rule pruning procedure is called. The basic idea is to iteratively remove one-term-at-a-time from the rule while this process improves the quality of the rule. More precisely, in the first iteration one starts with the full rule. Then it is tentatively tried to remove each of the terms of the rule – each one in turn – and the quality of the resulting rule is computed using a given rule-quality function (defined by Equation (5)). It should be noted that this step might involve replacing the class in the rule consequent, since the majority class in the cases covered by the pruned rule can be different from the majority class in the cases covered by the original rule. The term whose removal most improves the quality of the rule is effectively removed from it, completing the first iteration. In the next iteration it is removed again the term whose removal most improves the quality of the rule, and so on. This process is repeated until the rule has just one term or until there is no term whose removal will improve the quality of the rule.

3.4 Pheromone Updating

At each iteration of the WHILE loop of Algorithm I all $term_{ij}$, $\forall i, j$, are initialized with the same amount of pheromone, so that when the first ant starts its search, all paths have the same amount of pheromone. The initial amount of pheromone deposited at each path position is inversely proportional to the number of values of all attributes, and is defined by Equation (4):

$$\tau_{ij}(t=0) = \frac{1}{\sum_{i=1}^{a} b_i} \tag{4}$$

where:

- a is the total number of attributes,
- b_i is the number of possible values that can be taken on by attribute A_i.

The value returned by this equation is normalized to facilitate its use in Equation (1). The amount of pheromone associated with each $term_{ij}$ occurring in the rule found by the ant (after pruning) is increased in proportion to the quality – Q – of that rule. and is computed by the formula: $Q = sensitivity \times specificity$ [12], defined as:

$$Q = \frac{TP}{TP + FN} \cdot \frac{TN}{FP + TN} \tag{5}$$

where: TP (true positives) is the number of cases covered by the rule that have the class predicted by the rule; FP (false positives) is the number of cases covered by the rule that have a class different from the class predicted by the rule; FN (false negatives) is the number of cases that are not covered by the rule but that have the class predicted by the rule; and TN (true negatives) is the number of cases that are not covered by the rule and that do not have the class predicted by the rule.

Q´s value is within the range $0 \le Q \le 1$ and, the larger the value of Q, the higher the quality of the rule. Pheromone updating for a $term_{ij}$ is performed according to Equation (6), for all terms $term_{ij}$ that occur in the rule.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t) \cdot Q, \ \forall i,j \in R \tag{6}$$

where R is the set of terms occurring in the rule constructed by the ant at time t.

In Ant-Miner, the pheromone evaporation is implemented in a somewhat indirect way. More precisely, the effect of pheromone evaporation for unused terms is achieved by normalizing the value of each pheromone τ_{ij}. This normalization is performed by dividing the value of each τ_{ij} by the summation of all τ_{ij}, $\forall i,j$.

In order to classify a new test case, unseen during training, the discovered rules are applied in the order they were discovered (recall that discovered rules are kept in an ordered list). The first rule that covers the new case is applied – this is, the case is assigned the class predicted by that rule's consequent. It is possible that no rule of the list covers the new case. In this situation the new case is classified by a default rule that simply predicts the majority class in the set of uncovered training cases, that is, the set of cases that are not covered by any discovered rule.

4 Computational Results and Discussion

4.1 Data Sets, Discretization Method and Parameters Values Used in the Experiments

The performance of Ant-Miner was evaluated using six public-domain data sets from the UCI (University of California at Irvine) repository[1].

[1] Available in the Internet in the address: http://www.ics.uci.edu/~mlearn/MLRepository.html.

The main characteristics of the data sets used in our experiment are summarized in Table 1. The first column of this table identifies the data set, and the other columns indicate, respectively, the number of cases, number of categorical attributes, number of continuous attributes, and number of classes of the data set.

Table 1. Data Sets Used in the Experiments

Data Set	#cases	#categ. attrib.	#contin. attrib.	#classes
Ljubljana breast cancer	282	9	-	2
Wisconsin breast cancer	683	-	9	2
tic-tac-toe	958	9	-	2
dermatology	366	33	1	6
hepatitis	155	13	6	2
Cleveland heart disease	303	8	5	5

As mentioned earlier, Ant-Miner discovers rules referring only to categorical attributes. Therefore, continuous attributes have to be discretized in a preprocessing step. This discretization was performed by the C4.5-Disc discretization method [11].

In all the experiments Ant-Miner's parameters values were set to: *No_of_ants* = 3000; *Min_cases_per_rule* = 10; *Max_uncovered_cases* = 10 and *No_Rules_Converg* = 10.

We have made no serious attempt to optimize the setting of these parameters; however, they have produced quite good results, as will be shown later. In the same way, CN2's parameters were not optimized as well, in order to make the comparison between the two algorithms more fair.

4.2 Comparing Ant-Miner with CN2

We have evaluated the performance of Ant-Miner by comparing it with CN2 [3], a well-known classification-rule discovery algorithm. In essence, CN2 searches for a rule list in an incremental fashion. It discovers one rule at a time. Each time it discovers a rule it adds that rule to the end of the list of discovered rules, removes the cases covered by that rule from the training set and calls again the procedure to discover another rule for the remaining training cases. Notice that this strategy is also used by Ant-Miner. In addition, both Ant-Miner and CN2 construct a rule by starting with an empty rule and incrementally add one term at a time to the rule.

However, the rule construction procedure is very different in the two algorithms. CN2 uses a beam search to construct a rule.

In CN2 there is no mechanism to allow the quality of a discovered rule to be used as a feedback for constructing other rules. This feedback (using the mechanism of pheromone) is the major characteristic of ACO algorithms, and can be considered the main difference between Ant-Miner and CN2. In addition, Ant-Miner performs a stochastic search, whereas CN2 performs a deterministic search. In data mining and machine learning terminology, one can say that both algorithms have the same representation bias (since they both discover an ordered rule list), but different search (or preference) biases.

The comparison was carried out across two criteria, namely the predictive accuracy of the discovered rule lists and their simplicity. Predictive accuracy was measured by a well-known 10-fold cross-validation procedure [14].

All the results were obtained using a Pentium II PC with clock rate of 333 MHz and 128 MB of main memory. Ant-Miner was developed in C language and it took about the same processing time as CN2 (on the order of seconds for each data set) to obtain the results. The results comparing the accuracy rate of Ant-Miner and CN2 are reported in Table 2. The numbers right after the "±" symbol are the standard deviations of the corresponding accuracy rates.

Table 2. Accuracy Rate of Ant-Miner vs CN2

Data Set	Ant-Miner's Accuracy rate (%)	CN2's Accuracy rate (%)
Ljubljana breast cancer	75.28 ± 2.24	67.69 ± 3.59
Wisconsin breast cancer	96.04 ± 0.93	94.88 ± 0.88
tic-tac-toe	73.04 ± 2.53	97.38 ± 0.52
dermatology	94.29 ± 1.20	90.38 ± 1.66
hepatitis	90.00 ± 3.11	90.00 ± 2.50
Cleveland heart disease	59.67 ± 2.50	57.48 ± 1.78

Concerning classification accuracy, Ant-Miner obtained results somewhat better than CN2 in four of the six data sets, whereas CN2 obtained a result much better than Ant-Miner in the Tic-tac-toe data set. In one data set both algorithms obtained the same accuracy rate. Therefore, overall one can say that the two algorithms are roughly competitive in terms of accuracy rate, even though the superiority of CN2 in the tic-tac-toe is more significant than the superiority of Ant-Miner in four data sets.

We now turn to the results concerning the simplicity of the discovered rule list. This simplicity was measured, as usual in the literature, by the number of discovered rules and the average number of terms (conditions) per rule. The results comparing the simplicity of the rule lists discovered by Ant-Miner and by CN2 are reported in Table 3.

Table 3. Simplicity of Rule Lists Discovered by Ant-Miner vs CN2

	No. of rules		No. of terms / No. of rules	
Data Set	Ant-Miner	CN2	Ant-Miner	CN2
Ljubljana breast cancer	7.10 ± 0.31	55.40 ± 2.07	1.28	2.21
Wisconsin breast cancer	6.20 ± 0.25	18.60 ± 0.45	1.97	2.39
Tic-tac-toe	8.50 ± 0.62	39.70 ± 2.52	1.18	2.90
Dermatology	7.30 ± 0.15	18.50 ± 0.47	3.16	2.47
Hepatitis	3.40 ± 0.16	7.20 ± 0.25	2.41	1.58
Cleveland heart disease	9.50 ± 0.92	42.40 ± 0.71	1.71	2.79

Concerning the simplicity of discovered rules, overall Ant-Miner discovered rule lists much simpler (smaller) than the rule lists discovered by CN2.

This seems a good trade-off, since in many data mining applications the simplicity of a rule list/set tends to be even more important than its accuracy rate.

5 Conclusions and Future Work

We have compared the performances of Ant-Miner and the well-known CN2 algorithm in six public domain data sets. The results showed that, concerning predictive accuracy, Ant-Miner obtained somewhat better results in four data sets, whereas CN2 obtained a considerably better result in one data set. In the other data set both algorithms obtained the same predictive accuracy. Therefore, overall one can say that Ant-Miner is roughly competitive with CN2 with respect to predictive accuracy.

On the other hand, Ant-Miner has consistently found much simpler (smaller) rule lists than CN2. Therefore, Ant-Miner seems particularly advantageous when it is important to minimize the number of discovered rules and rule terms (conditions), in order to improve the comprehensibility of the discovered knowledge. It can be argued that this point is important in many (probably most) data mining applications, where discovered knowledge will be shown to a human user as a support for intelligent decision making, as discussed in the introduction.

Two important directions for future research are as follows. First, it would be interesting to extend Ant-Miner to cope with continuous attributes as well, rather than requiring that this kind of attribute be discretized in a preprocessing step. Second, it would be interesting to investigate the performance of other kinds of heuristic function and pheromone updating strategy.

References

[1] E. Bonabeau, M. Dorigo and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. New York, NJ: Oxford University Press, 1999.

[2] L. A. Brewlow and D. W. Aha, "Simplifying decision trees: a survey," *The Knowledge Engineering Review*, vol. 12, no. 1, pp. 1-40, 1997.

[3] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine Learning*, vol. 3, pp. 261-283, 1989.

[4] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, New York: John Wiley & Sons, 1991.

[5] M. Dorigo, A. Colorni and V. Maniezzo, "The ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, vol. 26, no. 1, pp. 1-13, 1996.

[6] M. Dorigo and G. Di Caro, "The ant colony optimization meta-heuristic," In: *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover Eds. London, UK: McGraw Hill, pp. 11-32, 1999.

[7] M. Dorigo, G. Di Caro and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137-172, 1999.

[8] U. M. Fayyad, G. Piatetsky-Shapiro and P. Smyth, "From data mining to knowledge discovery: an overview," In: *Advances in Knowledge Discovery & Data Mining*, U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy Eds. Cambridge: AAAI/MIT, pp. 1-34, 1996.

[9] A. Freitas and S. H. Lavington, *Mining Very Large Databases with Parallel Processing*, London: Kluwer, 1998.

[10] A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*, (Forthcoming book) Heidelberg: Springer-Verlag, 2002.

[11] R. Kohavi and M. Sahami, "Error-based and entropy-based discretization of continuous features," In: *Proceedings of the 2nd International Conference Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press, pp. 114-119, 1996.

[12] H. S. Lopes, M. S. Coutinho and W. C. Lima, "An evolutionary approach to simulate cognitive feedback learning in medical domain," In: *Genetic Algorithms and Fuzzy Logic Systems*: *Soft Computing Perspectives*, E. Sanchez, T. Shibata and L.A. Zadeh Eds. Singapore: World Scientific, pp. 193-207, 1998.

[13] N. Monmarche, "On data clustering with artificial ants," In: *Data Mining with Evolutionary Algorithms*, Research Directions – Papers from the AAAI Workshop, Technical Report WS-99-06, A.A. Freitas Ed. Menlo Park: AAAI Press, pp. 23-26, 1999.

[14] S. M. Weiss and C. A. Kulikowski, *Computer Systems That Learn*, San Francisco, CA: Morgan Kaufmann, 1991.