Evolution of Tags in Classifier Systems

Araceli Sanchis,[*] José M. Molina, Pedro Isasi,[‡] Javier Segovia[1§]

Sca-Lab. Departamento de Informática, Universidad Carlos III de Madrid, Spain, Avda. Universidad 30, 28911-Leganés (Madrid); [1]Departamento de Lenguajes, Sistemas e Ingeniería del Software, Facultad de Informática, UPM, Campus de Montegancedo, Boadilla del Monte (Madrid)

ABSTRACT

One of the major problems related to Classifier Systems is the loss of rules. This loss is caused by the Genetic Algorithm being applied on the entire population of rules jointly. Obviously, the genetic operators discriminate rules by the strength value, such that evolution favors the generation of the stronger rules. When the learning process presents individual cases and allows the system to learn gradually from these cases, each learning interval with a set of individual cases can lead the strength to be distributed in favor of a given type of rules that would, in turn, be favored by the Genetic Algorithm. Basically, the idea is to divide rules into groups such that they are forced to remain in the system. This contribution is a method of learning that allows similar knowledge to be grouped. A field in which knowledge-based systems researchers have done a lot of work is concept classification and the relationships that are established between these concepts in the stage of knowledge conceptualization for later formalization. This job of classifying and searching relationships is performed in the proposed Classifier Systems by means of a mechanism, Tags, that allows the classification and the relationships to be discovered without the need for expert knowledge.

Reprint requests to: Dr. José M. Molina, Sca-Lab. Departamento de Informática. Universidad Carlos III de Madrid, Spain. Avda. Universidad 30, 28911-Leganés (Madrid). e-mail: molina@ia.uc3m.es

[*]masm@inf.uc3m.es.; [‡]isasi@ia.uc3m.es. [§]fsegovia@fi.upm.es.

KEY WORDS

evolutionary computation, learning classifier systems, rule based systems, knowledge acquisition, knowledge classification, internal tags

1. INTRODUCTION

Classifier Systems (Holland, 1992, 1980, 1985, 1986a, 1986b, 1995) (Lanzi, et al., 2000), the subject of this paper, are studied from the viewpoint of behavior, an approach that considers exclusively the change in system behavior and is defended, among others, by Narendra, Thathachar, and Simon (Thathachar & Narendra, 1989).

Classifier Systems (CS) combine the advantages of rule-based systems with the possibility of applying a domain-independent learning system, such as Genetic Algorithms. The relative value of the different rules is one of the key information items to be learned in a CS. To promote this learning, CS oblige the rules to coexist in what is called an information-based economy service. Rules are made to compete, where the right to respond to the activation flows from the highest bidders, which will pay the value of their bids to the rules that are responsible for their activation. A chain of intermediaries is formed along this path, ranging from manufacturers (detectors) to consumers (actions to the environment). The competitiveness of the economy assures that the good (beneficial) rules survive and the bad ones disappear. There is a high level of relation and communication between the different levels of a CS (Golberg, 1989).

The conditions and messages of a CS form a system of rules, making them a special class of production system. One of the main problems raised by production systems is the complexity of rule syntax. CS find a way around this problem by restricting each rule to a fixed-length representation. This constraint has two benefits: first, all the rules, within a permitted alphabet, are syntactically meaningful and, second, a representation using fixed-length strings allows the application of genetic-type string operators. This opens the door to search of the space of permitted rules using Genetic Algorithms (Dumitrescu et al., 2000).

As discussed above, traditional Classifier Systems combine rule-based knowledge representation with genetic learning. There is an obvious difference between systems that use Genetic Algorithms for learning and Classifier Systems. In the former, the solution to the problem is fully encoded in the binary representation used by the Genetic Algorithm, that is, the evaluation of one individual is tantamount to the evaluation of the whole solution (Mitchel, 1996). In Classifier Systems, however, the evaluation of an output is equivalent to the evaluation of a rule that partly contributes to solving the problem. This evaluation is distributed across all those rules that contribute to the activation of the end rule, using the credit reassignment algorithm (Golberg, 1989). In no case, however, is it an evaluation of the system composed of all the rules. This is the approach proposed by the University of Michigan (Holland, 1986b). New rules or sets of rules are generated from these evaluations. So, any rules that have been activated and provide a satisfactory solution to part of the problem will be the source of new rules. The way in which Classifier Systems operate has some drawbacks, of which the following deserve a special mention:

- With regard to the system's ability to learn chains of rules that, moreover, do not break from one learning instant to another; the loss of a rule from the chain can lead to a loss of all the knowledge due to the interrelations between rules. The rules make sense not individually but only as groups that are unknown *a priori*.
- With regard to the need to apply the discovery algorithm to generate increasingly better classifiers and, finally,
- With regard to the sequencing of the cases put to the system to guide learning towards an improvement in overall system behavior.

The problem addressed in this paper is in particular how to combat the problem of the loss of rules and the need to 'maintain acquired knowledge'. Both problems are due to the application of Genetic Algorithms in CS, which leads the mechanisms of the CS to fail when forming and maintaining associations among rules. The Genetic Algorithm acts on the set of classifiers that have just been executed in such a manner that the new rules are generated from the best rules before discovery level action. This operation can lead to the loss of rules that are necessary for solving certain points of the problem and that appeared at the start of the learning period but failed to do so later on. This means that rules that were very good at the start of the execution can be considered by the GA as less valuable, because other rules are stronger. "Internal Tags" (IT), proposed by Holland (Holland, 1995) and others for application to Genetic Algorithms, were introduced for this purpose, giving rise to a new class of CS, Classifier Systems with Tags (TCS). Besides from preventing the loss of rules, different rules must be made to coexist at all times, thus stopping the rules becoming uniform, leading to a loss of variety in the rule population.

Classifier System performance is described in Sec. 2 and the related works that address the problem of loss of rules in CS. Section 3 contains the proposed system, the TCS. In Sec. 4, the experimental environment is presented. Results and comparison between CS and TCS are shown in Sec. 5. Learned rules of TCS are analyzed in Sec. 6. Finally, some conclusions are included.

2. CLASSIFIER SYSTEMS AND RELATED PROBLEMS

A Classifier System is composed of three main components, which can be considered as activity levels. The first level (Performance Level) is responsible for giving responses (satisfactory or otherwise) to solve the problem proposed. At this level, there are system rules, encoded by means of restricted alphabet character strings. When this level is executed, a response is given to a particular situation. The fitness of the response to the problem that is to be solved is measured by means of the reward received by the above rule from the environment. The second level (Credit Assignment) evaluates the results obtained at the lower level, distributing the rewards received by the rules that provide the output among all those that contributed to activating each of the latter rules. As this is a reinforced learning method, this evaluation can be adjusted by applying a reward or payment by the environment, whose value will be high if the solution is satisfactory and low if it is not. Reassignment can be carried out by means of different algorithms (Holland, 1986a) (Liepins et al., 1991), of which the Bucket Brigade (Holland, 1985) is the most commonly used and the one employed in this paper. At this level, it is not possible to modify system behavior by changing its rules; however, it is

possible to adjust their values and establish some sort of hierarchy of good and bad rules. The mission of the third level (Discovery) is to find new means for the system to discover new solutions, for which purpose a Genetic Algorithm (GA) is used.

Basically, the problem with discovery level action is that all the rules are considered to be equal. This idea, logical in other Evolutionary Computation techniques, where each individual is a solution to the problem, and they, therefore, all have to compete with each other, is not directly extendible to CS. This is because no one rule is capable of solving the problem on its own in many cases, which means that not all the rules are equal. A rule that is fired in a particular situation and whose action solves the problem is not the same as a set of rules that must be fired in order to address a different situation. Here, the strength of the first rule is likely to grow much more than the strength of all the rules chained in the second case. Furthermore, the distribution of the payment among members of one family means that the knowledge acquired earlier is not so quickly forgotten, as a rule that attains a given strength value continues to receive strength as a result of the execution of rules belonging to its family. The loss of rules is especially critical when the problem that is to be solved requires complex rule chaining, as the loss of a rule in the chain at the discovery level can mean that all the chaining is overlooked and the chain is entirely forgotten, which will mean that it will have to be learned again later.

2.1 Ad-hoc Internal CS Hierarchies

The problems of rule loss are addressed from various viewpoints in the literature with a view, in all cases, to improving CS. Shu and colleagues (Shu & Schaffer, 1991) consider introducing hierarchies into CS, that is, groups of rules that have to be maintained throughout the learning process. The rule groups are formed *a priori* and are given by the expert problem-solver. This is an attempt to solve the problem that DeJong (Booker et al., 1989) solved by means of *crowding* in the field of Genetic Algorithms. So, on the one hand, they establish rule groups (families) and, on the other, they propose genetic operators that act intrafamily and interfamily. The payment system is also modified, and when a rule from one group wins, all the other rules in its group also partake of that prize.

2.2 Hierarchically Organized Independent CS

In 1995, Dorigo (1995) presented the results of solutions designed to make Classifier Systems learn faster. The tools he used are parallelism, a distributed architecture, and training. With respect to parallelism and the parallel architecture, he proposes a parallel version of ICS (Dorigo & Schnepf, 1993), and designed a parallel Classifier System, called *Alecsys*, applied to what is termed the 'animat problem' (Wilson, 1985). This problem is addressed from the viewpoint of dividing the problem into smaller parts, based on a hierarchical architecture in which a series of ICS learn to cooperate in solving the learning problem. The different ICS levels are executed in parallel on different machines, and, moreover, different ICS, responsible for different tasks, are also executed in parallel. The author (Dorigo, 1995) takes up Brooks' (1991) idea of 'reactivity', that is, the existence of a set of behaviors, each of which is implemented by means of an ICS and which are independent of each other and produce an output for each input. The whole system is composed of three systems: an ICS to overcome obstacles, another to attain a goal and, finally, a system that decides which of the two possible outputs is the output of the combined system. The author proposes that internal conditions be included to achieve rule chaining (which is equivalent to behavior chaining in this case). This allows messages from the environment to be distinguished from messages from earlier cycles. Dorigo's study centers on the usefulness of the internal conditions without clearly explaining how they are used internally by the CS. The results of this part of the paper show that the size of these internal conditions, as applied in this case, is not very relevant for learning.

2.3 Limitations of CS Hierarchies

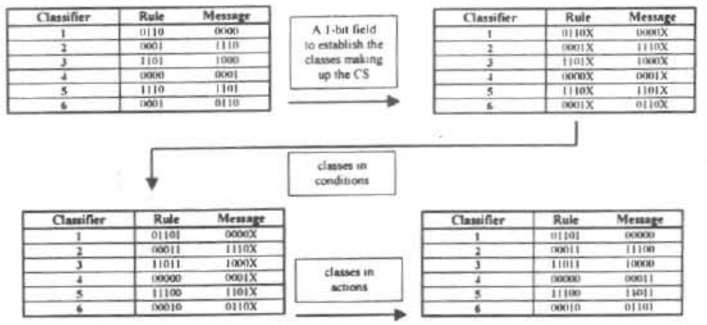
Shu (Shu & Schaffer, 1991) proposes dividing the CS rule set into subsets, each of which has rules specialized in a particular point of the problem, in such a manner as to make the members of the same family of rules compete. Dorigo's paper (1995) proposes a sort of hierarchy, since the final CS is composed of three CS: two basic CS and another that decides which CS is appropriate for each situation. In each case, rules are evolved independently,

in such a manner that each behavior evolves separately. The problem with this hierarchical approach as compared with Shu's proposal is that it is impossible to perform genetic operations that allow holistic evolution, as each Classifier System is evolved independently and is unrelated to the others. That is, no relationships can evolve between behaviors such that a rule from one classifier can activate a rule from another. The question is whether the separation of the Classifier System into several Classifier Systems raises system effectiveness in particular situations. In any case, it prevents the generalization of learning.

Automatic category generation within a CS has not been addressed in any paper to date. The idea can perhaps be borrowed from nature: some species use 'tags' to limit a 'call or warning' to a set of individuals, discriminating a subset among the total set. In the same manner, parts can be included in rules that allow some to be discriminated from others. What we will call *Internal Tags* (IT) can be defined in an *ad hoc* manner by creating a given string of calls (Shu & Schaffer, 1991) or can be defined in such a manner that the ITs themselves evolve, determining what groups are necessary. In short, each rule can be provided with a field that will evolve genetically and that identifies that the rule in question is a member of a group, similarly to the tags proposed by Holland (1995).

3. EVOLUTION OF TAGS IN A CS: THE TCS

As discussed in the preceding section, any solution that seeks to prevent the loss of rules necessarily involves creating subsets within the set of classifiers of which the CS is composed. In this work, the proposed solution must, therefore, combine the ability to learn without *a priori* knowledge and the capability of generating some kind of internal subdivision within the CS to allow categories of rules to exist. A CS, called TCS, has been designed that allows groups to evolve automatically. For this solution to be implemented, the encoding of the classifiers will have to be modified to include a field that represents the type or group to which each classifier belongs (see Fig. 1).



Classifier	Rule	Message
1	0110	0000
2	0001	1110
3	1101	1000
4	0000	0001
5	1110	1101
6	0001	0110

Classifier	Rule	Message
1	0110X	0000X
2	0001X	1110X
3	1101X	1000X
4	0000X	0001X
5	1110X	1101X
6	0001X	0110X

Classifier	Rule	Message
1	01101	0000X
2	00011	1110X
3	11011	1000X
4	00000	0001X
5	11100	1101X
6	00010	0110X

Classifier	Rule	Message
1	01101	00000
2	00011	11100
3	11011	10000
4	00000	00011
5	11100	11011
6	00010	01101

Fig. 1: Example of Tags in a Classifier System.

In Fig. 1 can be seen that a 1-bit field can be reserved to establish the classes making up the CS. This field can be used to subdivide the CS into several groups of classifiers, each of which contains the classifiers that have the same value in the new field. This field can be said to establish the classifier type or group. According to the definition of the value of the field that establishes the classes, there are 2 classes: one defined by classifiers whose value is 1 (classifiers 1, 2 and 3) and the other by those whose value is 0 (classifiers, 4, 5 and 6). Note that the definition of a class is determined by the value of the above field in the condition part of the rule, that is, rules that must have the same value in the field for activation are members of the same group. This field, which appears in the encoding, evolves in the same manner as the other fields, which means that the number and size of each class in the CS hierarchy is variable and must be learned. Wide ranging groups can be established, and all the classifiers could actually have the same value, in which case the system would operate like a classical CS. Apart from establishing the classifier type according to the value of the condition part, as it is included in the message part that evolves similarly, not only are the rule groups evolving, so is the form of intergroup activation. In this case, the group 1 classifiers activate group 0 classifiers and vice versa. Obviously, the CS must learn this type of activation and there are a lot of possible configurations.

Finally, it is important to take into account that the inclusion of a field in the classifiers means that a value must also be entered in the input message in the above position. This value is not determined by the environment; it is defined *a priori* by means of a value encoding the fact that the message in question is the environmental message. In this manner, the CS will have to learn which rule group having the same group definition field value is to be activated in response to the environmental message.

The appearance of hierarchies in the CS is subject to the information about the category to which the rule belongs being maintained in each rule. This information must evolve genetically; obviously, if the information about the category in each rule is capable of representing "n" different categories, the solution to the problem could be composed of m ($m<n$) categories and the remaining categories would be irrelevant. If this information is represented in each rule and it is allowed to evolve, the number of rules associated with a particular category is also variable; in this respect, the genetic evolution of the categories will not only allow the categories required to evolve but also for each one to have the size required to solve the problem.

The mechanism of including Internal Tags (IT) in rules is beneficial for evolving complex solutions within a CS. As the TCS is executed in parallel and all the rules are activated at the same time, a range of complex strategies are generated in the messages list by chaining rules from different groups. These strategies are maintained during the internal CS execution cycles and the best are learned by means of credit reassignment and discovery processes.

Apart from having to differentiate the encoding for different groups, another two levels of the CS will have to be adjusted: the credit reassignment algorithm (BBA) and the discovery algorithm (GA). This is due to the need for each rule group or hierarchy to gradually evolve in parallel. On the one hand, the credit earned by one rule needs to be distributed among all the rules of its group in order for these rules to beat other groups, in such a manner that the strength of each group can be considered as a factor to be taken into account when performing intergroup genetic operations. In this case, it is not only an individual that evolves; evolution is focused on the generation of compact groups, which are widely used and should, therefore, have a better rule set, without overlooking the need for groups whose elements, though perhaps fewer, are essential for developing the final strategy. Note that if the strength of all the rules of a group increases when one of the rules of the group is assessed as positive by the BBA, the strength of those groups of rules that are chained with this group will also be increased, as the percentage strength awarded for activation will be calculated on larger sums.

4. TSC EVALUATION IN THE GAME OF DRAUGHTS

In this paper, we seek to get a measure of the contribution of Internal Tags (IT) to the learning process in a Classifier System. A clear evaluation of the contribution of ITs in the encoding calls for a problem that is solved in a perfectly defined environment. The environment chosen in this case was the learning of draughts end games, that is, draughts matches where only a few pieces remain on the board at an advanced stage of the game.

The objective of applying the TCS to learning the game of draughts is not to obtain a CS that plays draughts; it is to apply Classifier Systems in a clear and defined environment that allows traditional Classifier Systems to be compared with the modification proposed in this paper, including IT. Obviously, there are a lot of systems that play draughts, some very successfully (Schaeffer, 1997). However, for the purposes of this study and comparison, a player following a random strategy will be used, and measurements will be taken of the games each type of CS (classical/with IT) wins against the random player using different configurations. In this paper, the opening boards are not used, as we work only with end games, where the maximum number of pieces is 5. These can be pieces of any kind and be situated in any valid position on the board.

Figure 2 shows the possible moves of a piece and a king on the board. The kings are crowned when a piece reaches opposite end of the board. The edges of the board are the limits of the moves. The edges of the board are not continuous. In this paper, the directions of the moves are considered absolute, as shown in Fig. 2.

When an opponent's piece is positioned in any of the directions in which a player's piece can be moved, the latter will take the piece that is in its path, by

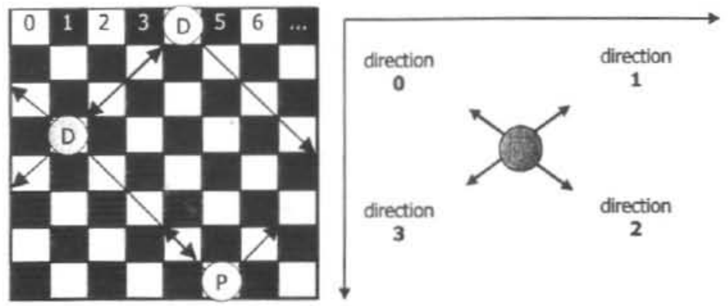


Fig. 2: Possible moves of pieces and kings on a board and definition of the directions of moves.

jumping over it onto the next vacant square in that direction. The piece captured will be removed from the board. This process will be repeated as many times as possible before the opponent player can take its turn. When either player has made a move or taken a piece (and cannot capture another piece), it will be the opponent's turn. The game will end when only one player's pieces remain on the board or there is a draw. There is a draw when the player whose turn it is cannot make any move.

4.1 Information Encoding

This involves analyzing how and what information about the board, the pieces, players, turns, moves, etc., can be supplied to the CS as an input message. The encoding chosen for the game of draughts is such that an output from the CS is always interpreted as a move. This means that the CS decisions are interpreted depending on the system status. Obviously, the system must be able to play with both black and white pieces, so an encoding was chosen that does not take into account 'the color' of the piece. Additionally, the directions of the moves have been taken to be absolute as explained above.

4.1.1 Input message. The information that is available on the board and that can be entered into the system is as follows:

- the number pieces on the board,
- the color of each piece,
- coordinates (x,y) of each piece, taken as shown in Fig. 2,
- piece type (piece or king),
- directions in which it can move or take and
- how far it can move or take in each direction.

The input messages include the status of the board at any one time: total number of pieces, number of pieces belonging to the CS player, color, whose turn it is, how many kings there are, etc. This information will be encoded in a 57-bit length input message for the traditional CS. The number of bits will be 61 for a classifier with IT, as 4 bits are entered to represent the ITs.

The first position of the input message encodes the information about the possibility of taking (with a 1) or only moving (with a 0). The next 4 positions contain information about the total number of pieces there are on the board, and the next 12 on the pieces that belong to the player whose turn it is, how many of these are kings and the number of the opponent's kings, all encoded using 4 bits in each case, considering the percentage represented. Then, the information regarding the position of these pieces is recorded, by transforming this decimal number into a binary number of up to 8 bits. Finally, if the total number of pieces is under 5, the remainder of the message is filled in with "#" symbols. The following are the steps (which are summarized in Fig. 2) taken for encoding an example set out in Fig. 3.

Figure 3 shows how the environmental data are transformed for entry into the CS input message. This conforms with the situation shown in Fig. 3, where the aim is to decide the move to be made by the white pieces.

4.1.2 Output message. The output message has the same length as the input message, 61 or 57 bits, depending on whether or not the ITs are taken into account. Only the last 16 bits of the entire message sent through the output interface of the CS after having performed the chaining process for several internal cycles are used as an output. A possible specimen output message is shown in Fig. 5.

	❶ Takes	❷ Total	❸ CS D.	❹ CS Kings	❺ Opponent K.	Dec. D1	Unid. D1	Dec. D2	Unid. D2	Dec. D3	Unid. D3	Dec. D4	Unid. D4	Dec. D5	Unid. D5
Number		3	2	1	1	4	0	5	7	1	3				
Percentage		6	4	2	2										
Binary	0	0110	0100	0010	0010	0100	0000	0101	0111	0001	0011	####	####	####	####

Fig. 3: Input message and board information encoding process in the TCS input interface.

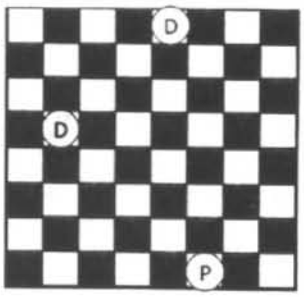


Fig. 4: Example of an end game.

Positions:	0		44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Message:	#	...	#	0	0	0	0	0	1	0	0	0	1	1	0	0	1	0	1

Fig. 5: Example of output message that should be translated into the appropriate action by the output interface.

With regard to the output messages, the respective positions will be taken and the decoding process will be performed. For ease of understanding, the specimen board shown in Fig. 3 will be detailed using the output message shown in Fig. 5.

1. Calculation of the piece to be moved/captured:

- The bits in positions 45 to 52 will be used to calculate which piece is to be moved/captured. In the example shown in Fig. 5, this is the binary number 000000100, which is equivalent to the decimal number 4.
- To decide to which piece it corresponds, the above decimal number is counted on the pieces actually on the board, and if they run out before reaching the above number, counting starts again from piece 0 to 4, inclusive; in the example, this would be 0,1,0,1,0, which means that the chosen piece would be 0, whose coordinates are (4,0).

2. Calculation of the direction of the move:

- The bits in positions 53 to 56 are used to calculate the direction in which the piece is to be moved. In the example, this is the binary number 0110, which corresponds to direction 6.
- To transform direction 6 into an actual move direction, the actual decimal direction is counted six times across the possible directions in which the piece can move, and when there are no real directions left, counting starts again from the first actual direction. In the example, piece 0 can move in only two directions, 2 and 3 (see Figure 4) (as it is a king that is on the edge of the board), which means that by counting from 0 to 6, inclusive, across the two possible directions, we will get 2,3,2,3,2,3,2, that is, the chosen direction will be 2.

3. Calculation of the positions that the piece is to be moved in the respective direction:

- The bits in positions 57 to 60 are used to calculate the number of squares to be advanced. In the example, this is the binary number 0101, which corresponds to the decimal number 5.

Now the above value has to be transformed into a real amount. For this purpose, if the value corresponds to a possible situation, the above value is taken; otherwise, it is transformed into a value between 1 and the

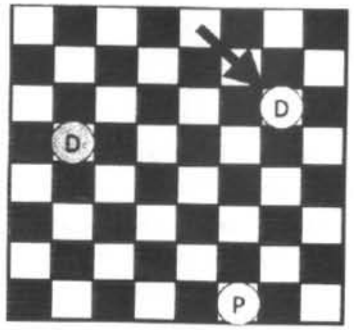


Fig. 6: Board in Figure 4 after having made the moves suggested by the TCS.

- maximum. In the example, 5 was output as the moves to be made and, yet, a move of only one, two or three squares is possible and, therefore, the piece is moved two squares.

In short, following the procedure detailed for the example, the strategy suggested by the TCS by means of the output message shown in Fig. 5 is to move the piece with coordinates (4,0) 2 squares in direction 2, which would give rise to the board shown in Fig. 6.

4.3 Payment Function

The objective of the payment function that analyzes the quality of the classifiers is to guide CS learning. The CS will learn depending on the payment function, and this, precisely, is the central objective of the CS developer when the CS are applied to a particular problem. For the purposes of this paper, the CS should be able to beat a random player, that is, a player who has no strategy and whose moves are not determined by the situation. This objective means that the payment function does not have to be able to evaluate different situations and detect the finest distinction in the moves decided by the CS; on the contrary, the payment function should be simple and objectively evaluate the decisions made by the CS, assessing each move on the basis of 'quantitative' results. In this manner, the CS will be able to beat a random, though not an experienced, player, and it will be possible to compare the classic CS and the TCS objectively.

The payment function takes into account the following factors: whether or not a piece has been captured, whether or not a king has been crowned, and the number of pieces taken. The payment will be made once the opponent has made a move. So, the payment function employed is based on the results achieved by the opponent and the results obtained by the CS move. In this case, payment can be represented by means of Table 1, where n_{cs} is the number of pieces taken by the CS and n_{op} the pieces captured by the opponent.

At the end of the game, the opponent player makes no move on the basis of which to evaluate the preceding move by the CS, so the result of the game is evaluated directly:

IF (the game ends in a draw) **THEN** (the payment is 400)

IF (the game does not end in a draw and the CS wins) **THEN** (the payment is 700)

IF (the game does not end in a draw and the opponent player wins) **THEN** (the payment is -700)

The payment developed is totally objective and depends on whether or not one piece is taken and on whether or not a king is crowned. This means that no payment is made if the move did not have a quantifiable result. Indeed, if there is no measurable quantity, the payment is 0. This payment 0 defines situations that will not be evaluated and, therefore, limits Classifier System learning ability. This limitation rules out any subjectivity coming into the payment that assesses CS operation, thus distorting the comparison between the classical CS and the TCS.

5. COMPARISON BETWEEN TRADITIONAL CS AND TCS

The objective of this section is to compare the traditional CS with the TSC. For this purpose, the above systems will be played against a player who makes random moves, having a variable degree of randomness and starting from different situations. The two systems commence without any previous knowledge

TABLE 1

Payment function for CS and TCS applied to the game of draughts

		CLASSIFIER SYSTEM			
		TAKES n_{CS} KING	Does not TAKE KING	TAKES n_{CS} No KING	Does not TAKE No KING
OPPONENT	TAKES n_{OP} KING	$n_{OP} \geq n_{CS}$: $-100*n_{OP}$	$-200*n_{OP}$	$n_{OP} \geq n_{CS}$: $-100*n_{OP}-10$	$-200*n_{OP}-10$
		$n_{OP} < n_{CS}$: $100*n_{OP}$		$n_{OP} < n_{CS}$: $100*n_{OP}-10$	
	Does not TAKE KING	$200*n_{CS}$	20	$200*n_{CS}-10$	-20
	TAKES n_{OP} No KING	$n_{OP} \geq n_{CS}$: $-100*n_{OP}+10$	$-200 * n_{OP} +10$	$n_{OP} \geq n_{CS}$: $-100*n_{OP}$	$-200*n_{OP}$
		$n_{OP} < n_{CS}$: $100*n_{OP}+10$		$n_{OP} < n_{CS}$: $100*n_{OP}$	
	Does not TAKE No KING	$200*n_{CS}+10$	70	$200*n_{CS}$	0

that is, their entire population is randomly generated, which means that their rules and messages are not adapted to any particular case and their moves will, in principle, also be random.

The three types of experiments conducted under this point were performed by gradually increasing their difficulty level to examine the behavior of the two systems in face of the above changes.

In the first type of experiments, the randomness of the random player is gradually raised. This means that there are different levels of randomness within a random player. This level of randomness is entered in the output message produced by the random player. The output message of the random player has the same make-up as that of the CS; however, it possesses only the sixteen characters required by the decoding process for transformation into a particular move. Randomness is entered in the output message of the random player depending on the number of characters are generated randomly. This generation is regulated proportionally, that is, there are random players whose output message is composed, for example, of 40% random characters.

Three groups of experiments with a different starting situation were performed for the comparison. The experiments were defined in increasing order of complexity, depending on the opening board with which each game that was to be played commenced:

1. First, the opening board will be fixed for all the games, then
2. the positions of the pieces that appear on the board in each game will be altered, and finally,
3. the opening board will be generated at random for each game.

In the first experiment, differing degrees of randomness will be applied to the opponent player, starting with 0% randomness and increasing this percentage up to 100% randomness. In the last two experiments, the opponent will 100% random throughout, and the opening boards will be modified incrementally, either by changing the position of the pieces or by generating a new board.

The result will show the evolution of the games won and lost by the two types of Classifier Systems. These results correspond to the average of five groups of games. In order to analyze the results obtained in more detail, the percentages of games won at the end of learning for each CS and for each

experiment, and the percentage improvement of the TCS as compared with the CS are set out in Table 2, Table 3, and Table 4. Analyzing the results, we find that the contribution of ITs to the CS is not relevant in all situations. In problems where the CS has to learn a very simple sequence of operations, because the problem to be solved is less complex, the ITs can turn out to be more of a handicap, as their inclusion means that the system is forced to "learn" how to chain rules, when such chaining may be unnecessary. As the problem becomes more complex, the need for rule chaining increases, and the contribution of the ITs becomes evident, since their existence encourages rule chaining. So, we find that the results of the TCS in the first experiments (Table 2) only improve on the CS in the last case. On the other hand, an improvement is seen in the results obtained with the TSC as compared with the CS in the subsequent experiments performed (Table 3 and Table 4).

Table 2 shows the results of the experiments in which the opening board was unchanged. In this case, the problem appears not to require rule chaining to develop strategies that can be used in unexpected situations, since the opening board is fixed and there are, therefore, only limited possibilities of different moves. So, the CS is faced with a player who, for all intents and purposes, makes a well-defined series of moves whose variability is very restricted. This is why the TCS results are 14% worse on average than those obtained by the CS. Considering that this is the simplest possible case, it appears that is counterproductive to force the CS to employ ITs, as it makes the TCS play worse than the CS. In the last case, where the systems face maximum variability, the results are very similar, and those obtained by the TCS are slightly better, mainly because the need for chained strategies starts to become evident.

Table 3 shows the results obtained when the opening board is modified using an incremental degree of randomness. In this case, the TCS performs 10% better on average than the CS; this is because the system has to start to generate more complex actions to be able to respond to more diverse situations. It is noteworthy in this case that the two systems obtain poor results at the maximum level of randomness, compared to the results that they obtained at lower levels of variability. This is perhaps due to the fact that these are very indeterminate situations where it is difficult for the system to be able to extract knowledge.

TABLE 2
Summary of the results of CS and TCS: Same Opening Board.

	0%	20%	30%	40%	50%	60%	70%	80%	90%	100%
CS	0.95	0.95	0.95	0.95	0.95	0.9	0.95	0.96	0.85	0.75
TCS	0.9	0.68	0.78	0.77	0.79	0.9	0.77	0.68	0.7	0.8
% Improvement	-5%	-27%	-17%	-18%	-16%	0%	-18%	-28%	-15%	5%

TABLE 3
Summary of CS and TCS results: Modified Opening Board.

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
CS	0.77	0.62	0.65	0.7	0.78	0.78	0.78	0.76	0.76	0.55
TCS	0.9	0.7	0.8	0.82	0.85	0.85	0.84	0.86	0.84	0.58
% Improvement	13%	8%	15%	12%	7%	7%	6%	10%	8%	3%

TABLE 4
Summary of CS and TCS results: Randomly Generated Opening Board.

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
CS	0.76	0.76	0.7	0.7	0.7	0.7	0.68	0.64	0.6	0.45
TCS	0.89	0.9	0.9	0.88	0.86	0.82	0.8	0.8	0.75	0.5
% Improvement	13%	14%	20%	18%	16%	12%	12%	16%	15%	5%

In Table 4, the results obtained show that as the degree of uncertainty in opponent player performance is increased, a higher percentage of the results of the TCS are better than those of the CS, in this case 15% on average. Again, neither of the two CS are able to obtain results of over 60% of games won with the effect of maximum randomness.

In short, we can infer from the results obtained that Classifier Systems are able to learn in games environments and that when the game is complicated, it requires a complex solution that is not satisfactorily provided by classical CS and thus requires the inclusion of tags. The results presented show how the proposed Classifier Systems are capable of improving on the classical approach of Classifier Systems in cases in which rule chaining is relevant. The importance of this contribution is the discovery of a learning method that allows similar or related knowledge to be grouped. This property of ITs, the automatic grouping of rules that share the same objective, is of special interest, and a study has, therefore, been conducted to analyze what effect they have and what results are obtained in each of the proposed Classifier Systems.

6. ANALYSIS OF THE GROUPS OF RULES LEARNED

The encoding used to represent the rules in the Classifier System used to play draughts employs many symbols. Not only is the number of symbols extensive, the encoding is very complex to ensure that all the outputs given by the Classifier System are valid. The need to generate valid responses at all times means that the meaning of the rule depends on the position of the pieces on the board. The meaning of the rules belonging to one group is, in this case, a problem for which there is no accurate analysis. Although the study of the meaning of the groups appears to be the best means of understanding what the Classifier System has learned, this is ruled out by the extent and complexity of the chosen encoding. Therefore, we will study the different situations in which the inclusion of the IT improves Classifier System learning. How the groups have evolved and the number of rules belonging to each group in the learning process is also of special importance.

To perform the experiments with the TCS, 4 bits were reserved in the condition part and message of each classifier. Eighty-one groups could be generated with 4 bits as shown in Table 5, each described by the value of its tags.

After analyzing the TCS obtained in the different experiments, we find that 10 groups are formed, whose tag values are as shown in Table 6. All the groups are formed using the alphabet {1,#}, which allows all the rules included in the TCS to be matched with messages that have the values "1111" in their IT area. This is because all the environmental messages have been identified with IT values "1111", and the TCS learns rules that can respond to these situations. However, not all the rules have the value "1111", which would mean that they only match with the environmental message, because the TCS learns the need to generate different groups and chain rules in internal cycles, where the environmental message is no longer involved.

The chosen encoding means that it is out of the question to interpret the classifiers obtained. Therefore, we will analyze how the number of rules in each of the groups formed changes as the experiment advances. The most complicated case of those shown was chosen. This corresponds with Table 4, that is, based on an opening board generated with 10% to 100% randomness. It is in this experiment that the inclusion of ITs in the CS improves most on the results obtained without ITs.

The evolution of the number of rules of the different groups are shown from Figure 7 to Figure 14. The figures show that the number of rules belonging to each group levels out as of the experiment with 70% randomness (Exp 70) in opening board generation. "1##1" and "###1" are the groups whose number of rules increase most. The groups with the most pronounced fall in the number of groups are: "111#", "1#11", "#111" and "####". Although they do undergo changes throughout the experiments, the number of rules in the other groups remain within a relatively narrow margin of values.

The evolution of the number of rules of the groups that increase and decrease most is related to the specificity of the IT values. The groups that decrease are more specific (except the one defined by "####" which will be dealt with separately) than those that increase; however, the three specific groups are included in the more general ones. In this manner, rules that belonged to the three groups and represented similar situations, even if they

TABLE 5

Tags for the possible groups that can be generated with 4 bits.

0000	1000	000#	00#0	0#00	#000	00##	0#0#	#00#	0###	####
0001	1001	001#	00#1	0#01	#001	01##	0#1#	#01#	1###	
0010	1010	010#	01#0	0#10	#010	10##	1#0#	#10#	#0##	
0011	1011	011#	01#1	0#11	#011	11##	1#1#	#11#	#1##	
0100	1100	100#	10#0	1#00	#100	0##0	#0#0	##00	##0#	
0101	1101	101#	10#1	1#01	#101	0##1	#0#1	##01	##1#	
0110	1110	110#	11#0	1#10	#110	1##0	#0#0	##10	###0	
0111	1111	111#	11#1	1#11	#111	1##1	#1#1	##11	###1	

TABLE 6

Tag values of the groups formed in the experiments with the TCS.

1111	111#	11#1	1#11	#111	11##	1##1	1###	###1	####

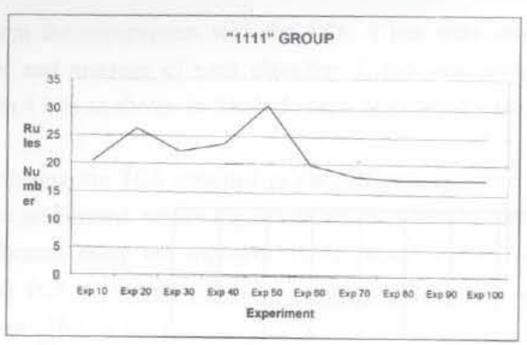


Fig. 7: Evolution in the number of rules of group "111" with increased board generation randomness.

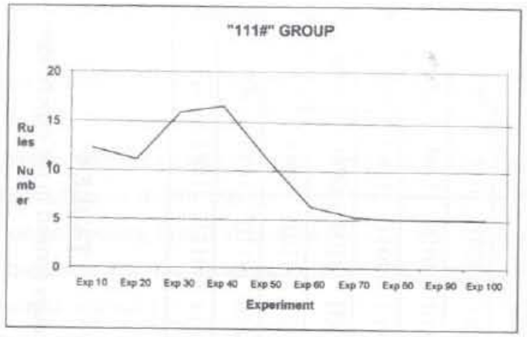


Fig. 8: Evolution in the number of rules of group "11#" with increased board generation randomness.

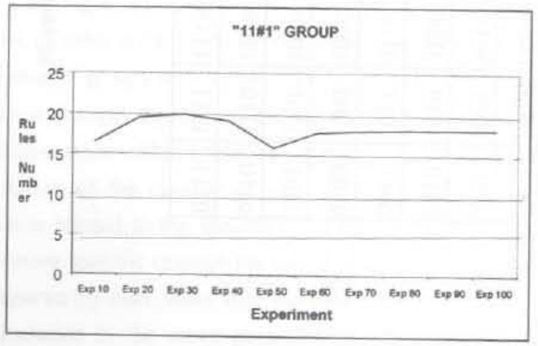


Fig. 9: Evolution in the number of rules of group "11#1" with increased board generation randomness.

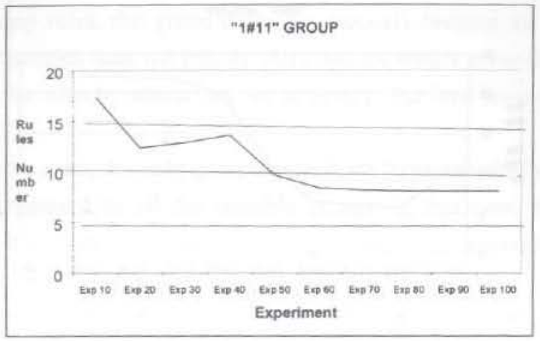


Fig. 10: Evolution in the number of rules of group "1#11" with increased board generation randomness.

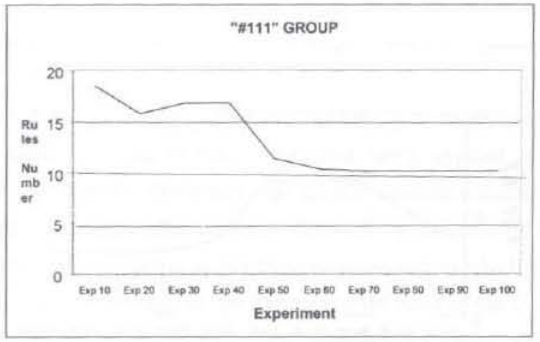


Fig. 11: Evolution in the number of rules of group "#111" with increased board generation randomness.

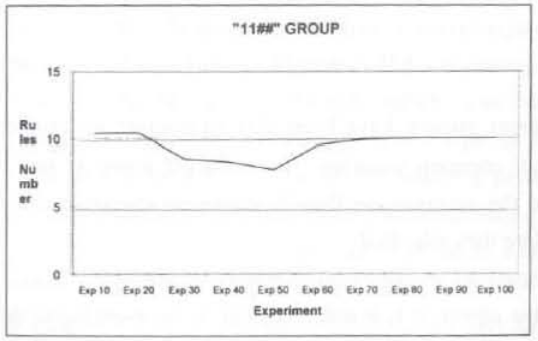


Fig. 12: Evolution in the number of rules of group "11##" with increased board generation randomness.

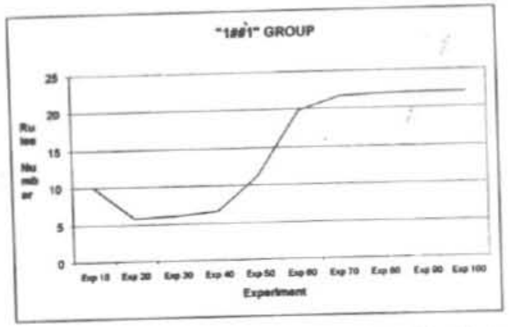


Fig. 13: Evolution in the number of rules of group "1##1" with increased board generation randomness.

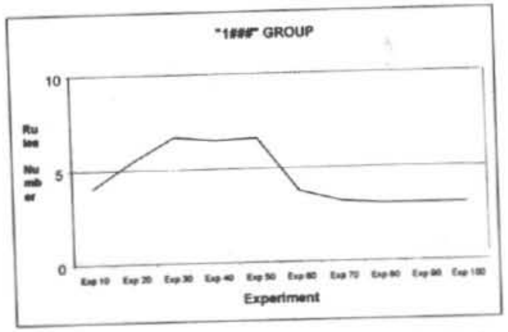


Fig. 14: Evolution in the number of rules of group "1###" with increased board generation randomness.

were in different groups, have been able to migrate to another group that represents that common situation. As the target group is less specific, the rules include the response to the IT values of the groups to which they belonged before they migrated.

With respect to group "####", which is the most general group and includes all the others, it is found that, first, in no experiment does it have a very significant number of rules (the maximum value is 7). Apart from not

containing many rules, this group decreases precisely because the generation of possible strategies does not require rules that are totally general and fail to discriminate the rules by which they are activated. The final leveling value of this group is 3.

As we have seen, the only groups formed are those required to solve the problem, as opposed to all the possible groups. In this case, this finding appears to be influenced by the IT values learned, as, although no more groups are necessary, these groups did have to act according to a particular hierarchy. In these experiments, we find that not only is the TCS able to learn groups of rules, it is also capable of establishing hierarchies between groups, using the "#" symbol.

CONCLUSIONS

One of the major problems related to Classifier Systems is the loss of rules. This loss is caused by the Genetic Algorithm being applied on the entire population of rules jointly. Obviously, the genetic operators discriminate rules by the strength value, such that evolution favors the generation of the stronger rules. When the learning system works in an environment in which it is possible to generate a complete training set, the strength of the rules of the CS will reflect the relative relationship between rules satisfactorily and, therefore, the application of the Genetic Algorithm will produce the desired effects. However, when the learning process presents individual cases and allows the system to learn gradually from these cases, each learning interval with a set of individual cases can lead the strength to be distributed in favor of a given type of rules that would in turn be favored by the Genetic Algorithm. If this reasoning is extended to the entire learning process, genetic diversity, which is so necessary for learning, can disappear due to the growth of a given type of rules in the population. Furthermore, when different rule sets are needed to solve part of the problem, these may disappear if part of the problem (corresponding to the rules that can be lost) is not presented in the examples found up to a certain point. However, the above rules can be very necessary.

This is an especially serious problem when there are very different rule types in the CS. Basically, the idea is to divide rules into groups such that they are forced to remain in the system; this allows groups of rules to survive by modifying the application of the classic genetic operators (mutation and crossover) at the discovery level and altering the payment function and the bids between rules of the same group, making the reward obtained by a rule of the group affect the whole group.

The objective of this paper was to obtain an encoding structure that would allow the genetic evolution of these groups in such a manner that their number and relationship would also be learned in the evolution process. For this purpose, an area that allows the definition of rule groups has been entered into the condition and message part of the encoded rules. This area will be named Internal Tags. This term was coined as the system has some similarities with natural processes that take place in certain animal species, where the existence of tags allows them to communicate and recognize each other.

This contribution is a method of learning that allows similar knowledge to be grouped. A field in which knowledge-based systems researchers have done a lot of work is concept classification and the relationships that are established between these concepts in the stage of knowledge conceptualization for later formalization (Gonzalez & Dankel, 1993). This job of classifying and searching relationships is performed in the proposed Classifier Systems by means of a mechanism that allows the classification and the relationships to be discovered without the need for expert knowledge.

REFERENCES

Booker, L., Goldberg, D.E. and Holland, J.H. 1989. Classifier systems and genetic algorithms, *Artificial Intelligence*, **45?**, 235–282.

Brooks, R.A., 1991. Intelligence without representation, *Artificial Intelligence*, **47**, 139–159.

Dorigo, M., 1995. ALECSYS and the autonomouse: learning to control a real robot by distributed classifier systems, *Machine Learning*, **19**, 209–240.

Dorigo, M. and Schnepf, U. 1993. Genetics-based machine learning and behavior based robotics: a new synthesis, *IEEE Trans. on Systems, Man and Cybernetics*, **23**, 141–154.

Dumitrescu, D., Lazzerini B., Jain L.C., Dumitrescu A. 2000. *Evolutionary computation*, CRC Press Series on Computational Intelligence.

Goldberg, D.E. 1989. *Genetic algorithms in search, optimization, and machine learning*, Reading, Massachusetts, USA, Addison Wesley.

González, A.J. and Dankel, D.D. 1993. *The engineering of knowledge-based systems*, New York, NY, USA, Prentice Hall.

Holland, J.H., 1980. Adaptive algorithms for discovering and using general patterns in growing knowledge bases, *International Journal of Policy Analysis and Information Systems*, **4**, 245–268.

Holland, J.H. 1985. Properties of the bucket brigade, in: *Proceedings of the International Conference on Genetic Algorithms and their Applications*, **1**, 1–7.

Holland, J.H. 1986a. A mathematical framework for studying learning in classifier systems, *Physica D*, **22**, 307–317.

Holland, J.H. 1986b. Escaping brittleness, the possibilities of general purpose learning algorithms applied to rule-based systems, in: *Machine Learning: An Artificial Intelligence Approach*, edited by Michalski, R.S, Carbonell, J.G. and Mitchell, T.M., CITY?, Morgan-Kaufman, 593-623.

Holland, J.H., 1995. *Hidden order: how adaptation builds complexity*, Reading, Massachusetts, USA, Addison-Wesley.

Holland, J.H., 1992. *Adaptation in natural and artificial systems*, Ann Arbor, Michigan USA, University of Michigan Press (First Edition, 1975).

Lanzi, P.L., Stolzmann, W. and Wilson, S.W. 2000. *Learning classifier systems from foundations to applications*, Lecture Notes in Computer Science, Springer.

Liepins, G.E., Hilliard, M.R., Palmer, M. and Ranjaran, G. 1991. Credit assignment and discovery in classifier systems, *International Journal of Intelligent Systems*, **6**, 55–69.

Mitchell, M., 1996. *An introduction to genetic algorithms*, Cambridge, Massachusetts, USA, MIT Press.

Schaeffer, J., 1997. *One jump ahead*, New York, Springer-Verlag.

Shu, L. and Schaeffer, J. 1991. HCS: Adding hierarchies to classifier systems, in: *Proceedings of the 4th International Conference on Genetic Algorithms*, 339-345.

Thathachar, M.A. and Narendra, K. 1989. *Learning automata, an introduction,* Englewood Cliffs, New Jersey, USA, Prentice Hall International.

Wilson S. 1985. Knowledge growth in an artificial animal, in: Proceedings of the First International Conference on Genetic Algorithms and their Applications, 16–23.