STUDIA INFORMATICA
Nr 1/2(7) Systemy i technologie informacyjne 2006

LCS and GP Approaches to Multiplexer's Problem

Katarzyna Wasielewska[1], Mariusz Bagiński[2], Franciszek Seredyński[3,4]
[1,2] The State Higher School of Vocational Education in Elblag,
The Institute of Applied Informatics
Wojska Polskiego 1, 82-300 Elblag, Poland,
[3] University of Podlasie, Institute of Computer Science
Sienkiewicza 51, 08-110 Siedlce, Poland
[4] Institute of Computer Science, Polish Academy of Science
Ordona 21, 01-237 Warsaw, Poland

Abstract. In this paper we present the use of learning classifier systems and genetic programming to solving multiplexer's problem. The function of multiplexer is the popular apparatus of researches which is used to investigate the effectiveness of systems based on evolutionary algorithms. It turns out that the eXtended Classifier System (XCS) learns the problem of multiplexer effectively and Genetic Programming (GP) finds the form of function of multiplexer correctly.

Keywords. Learning classifier system, genetic programming, multiplexer problem

1 Introduction

Learning Classifier System (LCS) is a rule-based learning machine introduced by John Holland [2]. This technique combines reinforcement learning and evolutionary computing to produce adaptive systems. LCS is the system in which rules (called *classifiers*) are generated with the use of Genetic Algorithm (GA). The GA operates on a population of classifiers.

Wilson introduced accuracy-based eXtended Classifier System (XCS) ten years ago [13]. The XCS consists of performance, reinforcement and discovery components. The reinforcement component uses Q-learning technique [10] to update classifiers. Whereas the discovery component consists in deleting classifiers from population and creating a new classifiers. The XCS makes the use of a niche mechanism and has tendency to evolve populations of maximally general classifier [13,7].

The Genetic Programming has been invented by N. L. Cramer [1] and popularized by J. Koza since 1994 year [4]. It is also treated as a form of adaptive teaching. The technique is based upon the GA which exploits the process of natural selection based on a fitness measure to create a population of solutions. These

solutions keep improving all the time. The GP uses encoding tree and also the same genetic operators like: selection, crossover and mutation. Each tree in population is a single computer program. Computer creates thousands of programs in one loop and it chooses the best fitness function.

The paper first gives an overview of the XCS and a short introduction to the GP. Section 4 presents results of experiments. Last section contains conclusions.

2 Overview of some techniques

2.1 Overview of the XCS

Traditionally, the XCS receives the message representing the current state of environment and executes the suitable action in environment. However, it does not have a message list and the *strength* parameter has a different form. And what is more, several new mechanisms appeared.

In single-step problem, at each time step the system receives the message from environment (see Figure 2.1) and the system compares this message with conditions of classifiers from population [P] and the system input creates a *match set* [M]. If the [M] is empty a new classifier is created through *covering* mechanism. Then for each possible action a_i the system prediction $P(a_i)$ is computed. The value $P(a_i)$ gives an evaluation of the expected reward if action a_i is performed. Next, there is action selection. The classifiers in [M] which propose the selected action are put in the *action set* [A]. The selected action is performed. Then an immediate reward is returned to the system. The reward is used to update the parameters of the classifiers in [A].

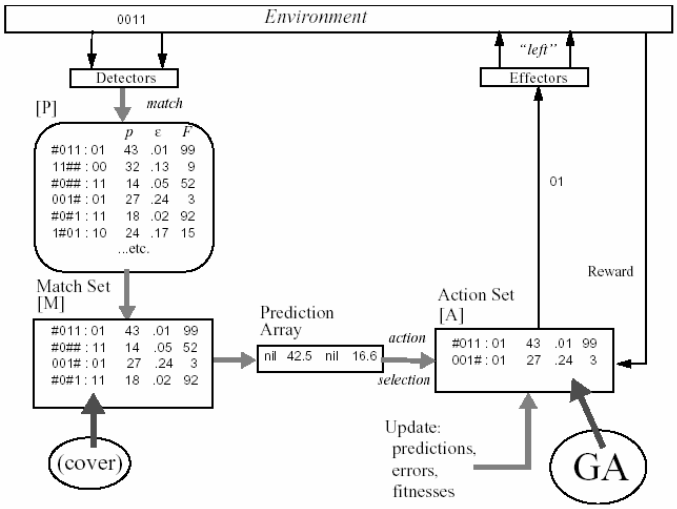

Figure 2.1. Schematic diagram of XCS for single-step problem (source: XCS tutorial, S. W. Wilson)

Each classifier in XCS has three parameters which are updated in each time: the prediction p_j, the prediction error ε_j and the fitness F_j. The parameter prediction p_j gives an estimate of what is the payoff P that the classifier is expected to gain. Reinforcement in XCS consists of updating these parameters. Classifier parameters are updated by the Widrow-Hoff delta rule [11]. It is updated according to $p_j \leftarrow p_j + \beta(P - p_j)$, where β is a learning rate constant $(0 < \beta \leq 1)$. The prediction error ε_j estimates what the exact the prediction p_j; it is updated according to $\varepsilon_j \leftarrow \varepsilon_j + \beta(|P - p_j| - \varepsilon_j)$. It applies these rules according to the moyenne adaptive modifee (MAM) technique [9]. However, the update of fitness has several phases. It is necessary to set the classification accuracy κ_j of each classifier as $\kappa_j = \alpha(\varepsilon_j / \varepsilon_0)^{-\nu}$ if $\varepsilon_j \geq \varepsilon_0$ or otherwise $\kappa_j = 1$, where α, ν are constants parameters accuracy function and $0 < \alpha < 1$ as well as ε_0 is a threshold such as, if the classifier's error is less than ε_0, the classifier gives accuracy 1. And the fitness parameter is updated by the rules $F_j \leftarrow F_j + \beta(\kappa_j' - F_j)$, where κ_j' is a *relative accuracy* which is calculated by dividing κ_j of each classifier in [A] by the sum of the κ_j s of the set. So, classifier fitness in XCS is based on the accuracy of the classifier prediction.

The genetic algorithm in XCS is applied to the action sets and acts in environmental niches. It consists in selecting two classifiers, copying them, and performing crossover and mutating on each allele.

Accuracy-based fitness and a niche GA cause that XCS's population tends to form a complete and accurate mapping X x A => P from inputs and actions to payoff predictions.

The XCS acts on so called *macroclassifiers*. These are classifiers that represent a set of classifiers with the same condition and the same action by means of a parameter called *numerosity*. Wilson shows that macroclassifiers are a programming technique that speeds up the learning process by reducing the number of real classifiers (*microclassifiers*) of XCS. The XCS evolves populations of accurate and maximally general classifiers. The description of generalization hypothesis may be found in [13].

The extension of the XCS are also new genetic operators: *subsumption deletion* [14] and *specify* [7]. *Subsumption deletion* acts when classifiers created by the GA are inserted in the population and thanks to its accurate classifier can produce only more general classifiers. Whereas *specify* assists the generalization mechanism to eliminate overly general classifiers. Lanzi shows that *specify* operator rather slows the generalization process [7].

2.2 Overview of GP

In Genetic Programming single computer program is presented most often as *S*-expression (LISP notation). For example let be a simple tree for multiplexer problem [4] presented in Figure 2.2. *S*-expression for this tree can be shown as OR(A1(NOT D0)).

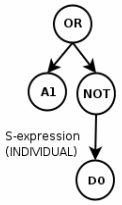

Figure 2.2. Example of *S*-expression (tree)

Somewhat otherwise like in GA, in GP we calculate fitness function. Generally are four cases of fitness function in GP [4]: (a) raw fitness $r(i,t) = \sum_{j=1}^{N} |S(i,j) - C(j)|$ of individual *S*-expression *i* in the population of size M at any generational time step *t,* where *S(i,j)* – the value returned by *S*-expression *i* for fitness case *j* (of *N* cases) and *C(j)* is the correct value for fitness case *j*; (b) standardized fitness $s(i,t) = r_{\max} - r(i,t)$ where r_{max} is a maximum possible value of raw fitness; in particular problem is $s(i,t) = r(i,t)$, e.g. in regression problem [5,6]; (c) adjusted fitness $a(i,j) = 1/(1+s(i,j))$ where $s(i,t)$ is the standardized fitness for individual *i* at time *t*; (d) normalized fitness $n(i,t) = a(i,t)/(\sum_{k=1}^{M} a(k,t))$ where M is force of population.

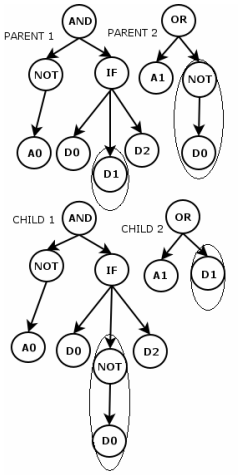

Figure 2.3. Example of crossover operator

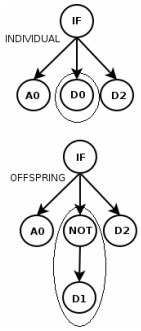

Figure 2.4. Example of mutation operator

In GP also occurs genetic operator like: crossover, mutation, reproduction (showed on Figure 2.3 and Figure 2.4).

Most important in GP is defining terminal *T* and function *F* sets. In our example of 6-multiplexer showed in Figure 2.2 we have T = {A0,A1,D0,D1,D2,D3} and F = {IF,OR,AND,NOT}. In T sets take a stand all variables defined within a problem. However, in the F set can take a stand all mathematics functions or specific operators properly defined for the problem and ADFs (*Automatically Defined Functions*) [3]. The ADF is a function (i.e. procedure, module) that is dynamically evolved during a run of GP and which may be called by a calling program that is simultaneously being evolved.

How GP works is presented on diagram in Figure 2.5.

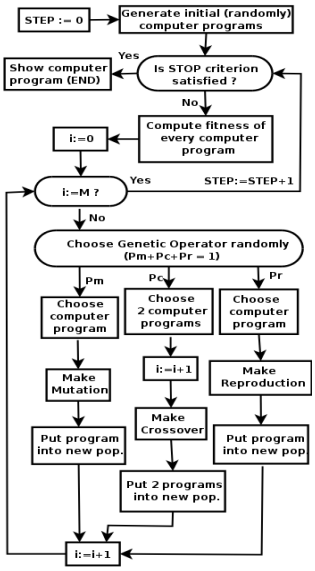

Figure 2.5. Idea of Genetic Programming

3 A multiplexer's function

The multiplexer's problem is a logical function commonly used for testing evolutionary techniques. The multiplexer's function is defined for strings of length $L=k+2^k$ where k is integer > 0. L-multiplexer has k address-inputs and 2^k data-inputs and one output always (0 or 1). For example for k=2 we have 6-multiplexer which has 2 address-inputs and 4 data-inputs. Many electronic devices act according to principle of multiplexer's function (example of representation in Figure 3.1). The multiplexer's problem is single-step problem, i. e. the choice of action does not affect future inputs.

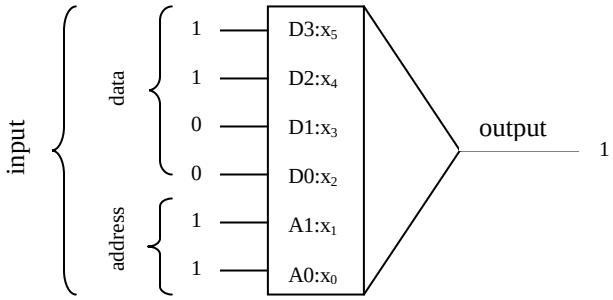

Figure 3.1. Schema of 6-multiplexer

In Figure 3.1 we see six inputs (110011) of which the first two (address-inputs) indexes one bit of data-inputs. The value of multiplexer function is the value of the indexed bit. In our example the value of 6-multiplexer function is 1, because bits of address – 11 – represent the last bit of data-inputs (i.e. D3). Similarly, the value of 000111 is 0.

The multiplexer function we can introduce in disjunctive normal form. For example, 6-multiplexer is $F_6 = x_0' x_1' x_2 + x_0' x_1 x_3 + x_0 x_1' x_4 + x_0 x_1 x_5$ where primes indicate negation.

4 Experiments with a multiplexer

In context of XCS, the multiplexer's problem means to learn a correct classification of input signals according to multiplexer's principles, whereas in context of GP – to find a logical function of multiplexer.

4.1 XCS approach to multiplexer's problem

This problem is encoded as a binary bitstring by the XCS's input interface and the system's action is returned by the output interface. We focus on 6-multiplexer. The input to the system consists of a string of 6 binary digits, of which the first two represent address into the remaining bits (the data). We receive

the 64-input space. The most specific classifier is 100101:0 and the most general is 10##0#:0. The classifier 10##0#:0 is correct for all inputs it can match. It is maximally general classifier. Notice, that we have the 8 maximally general classifiers (Figure 4.1).

000###:0	000###:0 10##0#:0
001###:1	000###:1 10##0#:1
01#0##:0	001###:0 10##1#:0
01#1##:1	001###:1 10##1#:1
10##0#:0	01#0##:0 11###0:0
10##1#:1	01#0##:1 11###0:1
11###0:0	01#1##:0 11###1:0
11###1:1	01#1##:1 11###1:1

Figure 4.1. The best action map (left) and the complete map (right) for 6-multiplexer

Taking into account 2 payoff levels (payoff p_1 for the correct answer and payoff p_2 for the wrong answer) we receive that the optimal population consists of the 16 maximally general classifiers (Figure 4.1). The XCS aspires to create a complete map of input/output. It means that XCS strives to create the population of classifiers which will be capable of matching themselves to each input and they will offer all possible actions.

The XCS has the task of executing the action which matches to input in the current situation in the environment. This is single-step problem.

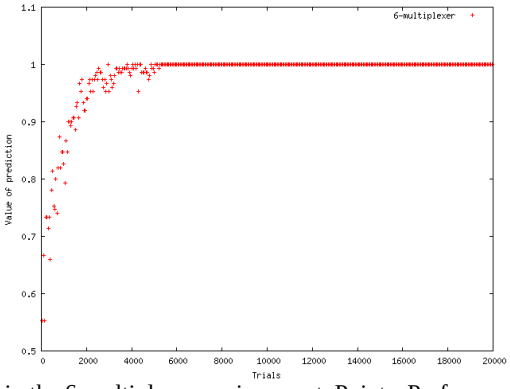

Figure 4.2. Results in the 6-multiplexer environment. Points: Performance. Population size in microclassifiers. Parameters (as [13]): N = 400, $\beta = 0.2$, $\gamma = 0.71$, $\theta = 25$, $\varepsilon_0 = 0.01$, $\alpha = 0.1$, $\chi = 0.8$, $\mu = 0.04$, $\delta = 0.1$, $\Phi = 0.5$, $P_{\#} = 0.33$, $p_I = 10.0$, $\varepsilon_I = 0.0$, $F_I = 10.0$

It is visible in the Figure 4.2, that XCS has done very well with multiplexer's problem. Quickly enough, XCS has started classifying the encoded messages. Many

scientists' experiments, for multiplexer's problem, have proved also the large effectiveness of XCS architecture.

Below we present the results of experiments with 11- and 20-multiplexer (Figure 4.3), where number of individuals in population is equal 790 and 2100 respectively. Let's notice, that together with an increase of problem's complexity, the time of learning increase.

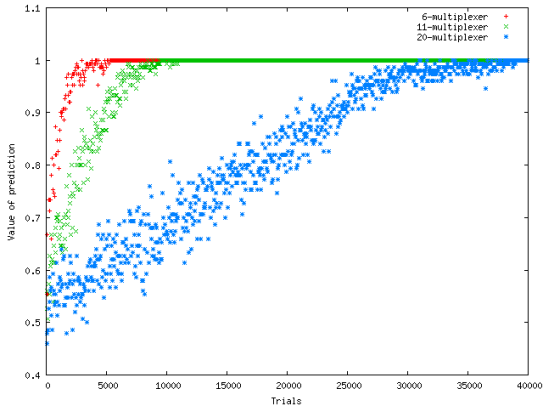

Figure 4.3. Comparison the results in the 6-, 11- and 20-multiplexer environment. Points: Performance. Population size in microclassifiers. Parameters: $N_6 = 400$, $N_{11} = 790$, $N_{20} = 2100$

It is worth mentioning, that Lanzi and Perruci have showed learning of the 6-multiplexer using XCS with *s-classifiers* [8], which are classifiers with Lisp s-expression conditions [12].

4.2 GP approach to multiplexer's problem

Solving problem of 6-multiplexer with GP, first of all we must define *T* and *F* sets. For 6-multiplexer we have *T* = {A0, A1, D0, D1, D2, D3} and logic functions set *F* = {IF, NOT, OR, AND}. Fitness function is equivalent to standardized fitness and equal 64 minus raw function. The GP technique representing number of hits is "a number of fitness cases for which the *S*-expression matches correct output" [4]. We want to obtain an *S*-expression which output is equal like 6-multiplexer. In experiment we use application *lilgp-1.1* [15].

All experiment was performed on machine with Ultra Sparc IIIi 1.5GHz processor with 1G RAM on Solaris 10 (Sun Blade 1500 workstation).

In first experiment we set: population size = 500, crossover = 0.8 (internal), crossover = 0.1 (external), mutation = 0.05 and reproduction = 0.05. After 44 sec. was obtained result in generation 40 in subpopulation 7. We obtained *S*-expression which in 100% correct describe 6-multiplexer (Adjusted fitness = 1 (Figure 4.4)).

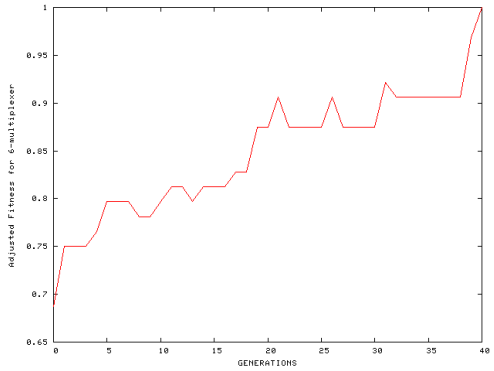

Figure 4.4. Adjusted fitness for 6-multiplexer. Population size = 500

Corresponding S-expression (computer program) is showed below:

(AND (NOT (NOT (OR (IF D2 A0 A0)(IF A1 D2 D0))))(OR (IF A1 D3 D1)(NOT (AND (IF (OR (NOT (AND A0 D2))(AND (IF A1 D3 D1)(AND (NOT D2)(AND D1 A1))))(OR (AND (OR (NOT D1)(AND (IF D0 D1 D2)(OR D0 D3)))(AND D3 A1))(IF A1 A0 A0))(IF (NOT A0)(OR A0 D0)(OR D1 A0))) (OR (NOT (IF A1 D3 D2))(AND (NOT (AND (IF (OR D3 D2)(AND D3 D1) (AND (IF D1 A0 A1)(IF (OR D3 D1) D3 A1)))(AND (AND A1 D0)(IF A1 D1 A1))))(IF (IF (OR (AND D2 D1)(OR A1 A1))(NOT (AND D3 D0))(AND (OR A1 D2)(OR A1 D2)))(OR (AND (IF D0 A1 D2)(IF D0 D1 D0))(OR (OR (AND D2 D0)(AND D1 D1))(NOT D1)))(OR (NOT (OR D0 D2))(AND (IF (NOT (IF D3 D2 A0)) (AND A0 D3)(AND (OR A1 D0)(OR A1 D2)))(IF A0 D0 D2)))))))))

Increasing population size to 2000 we obtained best solution after 48 sec. in generation 14 (subpopulation 7) (Figure 4.5)

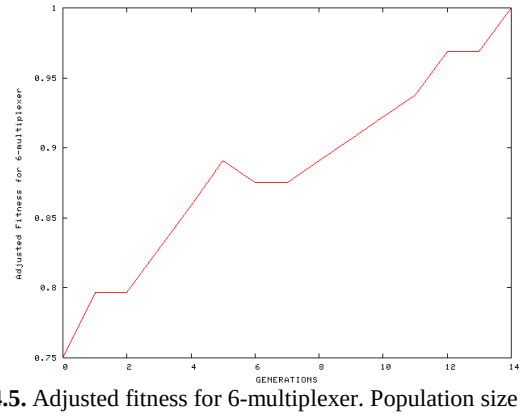

Figure 4.5. Adjusted fitness for 6-multiplexer. Population size = 2000

Computer program in LISP is showed below:

(IF (AND (OR D1 A1)(OR A1(AND A0 D3)))(AND (IF A0 D3 D2)(IF (AND A0 D1)(IF A0 D1 D2)(OR (OR (OR (AND A0 D3)(NOT A0))(OR (IF A1 D3 D0) D1))(OR (OR (IF D0 A0 D2)(OR D0 D0))(IF (IF D0 D0 D0)(AND D3 D2) (NOT D0))))))(IF (AND A1 A0)(OR D2 D2)(IF (NOT A0)(OR A0 D0)(IF A1 (AND D1(AND D3(AND D0(AND D2(NOT A0))))) D1))))

For k = 3 we have 11-multiplexer which has three address inputs and eight data inputs. For this case we have: T set: T = {A0, A1, A2, D0, D1, …, D7} and F set: F = {IF, NOT, OR, AND}. Fitness function is equivalent to standardized fitness and equal 2048 minus raw function.

We use the same value of genetic operation like in 6-multiplexer. In first case we have population size 500. After 24437 sec. we obtained the best solution in 346 generation (in subpopulation 4) (Figure 4.6).

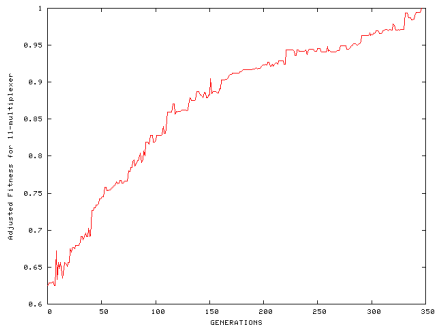

Figure 4.6. Adjusted fitness for 11-multiplexer. Population size = 500

Changing population size on 2000 we obtained next 100% solution. The solution was find after 23894 sec. in generation = 76 (subpopulation 6) (Figure 4.7).

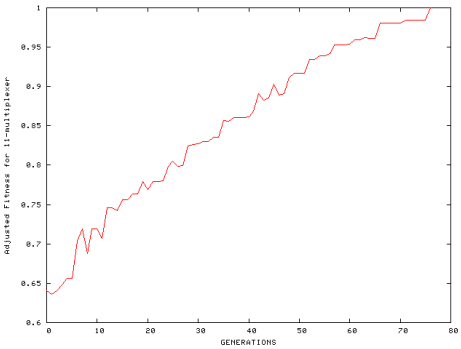

Figure 4.7. Adjusted fitness for 11-multiplexer. Population size = 2000

In 11-mulitplexer's problem we obtained very long *S*-expression and we cannot show this solutions in our publication. For example first solution consists of 503 nodes and tree has depth 16. Second solution has 441 nodes and depth 17.

5 Summary

The XCS and the GP solve the multiplexer's problem but in different manners. Setting other parameters we may obtain better or worse solution in a time. Wilson has claimed that comparison between XCS and GP in solving the 6-multiplexer indicates that speed of a learning process in XCS may be even thousand times faster [14].

The XCS evolves the population of classifiers that solve this problem. And the GP evolves individuals which are full solutions to multiplexer's problem. Whereas, the GP needs more time to find solution of problem and XCS is really quickly. However, the XCS system has considerably more complicated structure and it is more difficult for implementation.

References

1. Cramer, N. L. (1985). *A representation for the adaptive generation of simple sequential programs.* Proceedings of an International Conference on Genetic Algorithms and the Applications, Carnegie-Mellon University, Pittsburgh, PA, USA, 183-187.
2. Holland J. H., (1986). Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems, in: M. et al. (Ed.), *Machine learning, an artificial intelligence approach*. Volume II, Morgan Kaufmann.
3. Koza John R., (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*, Cambridge, Bradford Book / MIT Press 4 s. XXI, 746 Complex Adaptive Systems.
4. Koza J. R., (1994). *Genetic Programming: on the Programming of Computers by Means of Natural Selection,* Cambridge, Bradford Book / MIT Press 4 s. XV, 819 Complex Adaptive Systems.
5. Koza J. R., (1994). G*enetic Programming for Economic Modeling*, Stanford University, California USA.
6. Langdon W. B., Qureshi Adil, (2000). *Genetic Programming – Computer using "Natural Selection" to Generate Programs,* University Collage London.
7. Lanzi P. L., (1997). A Study of the Generalization Capabilities of XCS, in Back, T. (ed.), *Proc. of ICGA97 Conference*, Morgan Kaufmann, San Francisco, 418-425.
8. Lanzi P. L., Perruci A., (1999). Extending the representation of classifier conditions, part II: from messy coding to s-expressions, GECCO.

9. Venturini G., (1994). *Apprentissage Adaptatif et Apprentissage Supervise par Algorithme Genetique*, These de Docteur en Science (Informatique), Universite de Paris-Sud.
10. Watkins C. J. C. H., (1989). *Learning from Delayed Rewards*, Ph.D thesis, Cambridge University.
11. Widrow B., Hoff M., (1960). Adaptive switching circuits. In Western Electronic Show and Convention, Institute of Radio Engineers (now IEEE) vol. 4, 96-104.
12. Wilson S. W., (1994). ZCS: a zeroth level classifier system. *Evolutionary Computation*, 1(2): 1-18.
13. Wilson S. W., (1995). Classifier Fitness Based on Accuracy, *Evolutionary Computation*, 3(2), 149-175.
14. Wilson S. W., (1996). Generalization in the XCS classifier system, unpublished contribution to the ICML'96 Workshop on Evolutionary Computing and Machine Learning, http://prediction-dynamics.com.
15. Zonker D., Punch B., (1996). *Lil-gp 1.01 User's Manual*, Michigan State University.