Learning Classier Systems

Karsten Øster Lundqvist

LCS Background

- Derived from the evolutionary studies (e.g. Genetic Algorithms) of Holland in the mid-seventies. First described in
  - J. H. Holland. Progress in Theoretical Biology IV, chapter Adaptation, pages 263–293. Academic Press, 1976.
- This is only a short overview, so read more in your own time (Especially if you are on LCS...)
  - Introduction to Evolutionary Computing, chap 7
  - Clever Algorithms Nature-Inspired Programming Recipes, Lulu, section 3.9. (Includes a working ZCS in Ruby)
  - Learning Classifier Systems: A Brief Introduction, Larry Bull (on BB)
  - http://www.cs.bris.ac.uk/~kovacs/lcs/search.html

LCS Background

- Creating an adaptive system based on evolution
  - Suited for problems that have: Perpetually novel events with significant noise, continual real-time requirements for action, implicitly or inexactly defined goals, and sparse payoff or reinforcement obtainable only through long sequences of tasks. (Clever Algorithms)
- Cooperative population of sub-solutions
- ‘if < conditions > then < action >’ production rules within population
- Reinforcement learning

LCS System

- The state of the environment is transmitted to the system via a set of **detectors** whose output is put on a **message list**.
- This message list may also contain other signals posted there by rules that have fired on previous cycles, or the detector signals from previous cycles. This means it may act as a form of memory.
- The **condition part** of each rule in the rule base is then examined to see if it matches the current message list. Those rules that match are tagged as belonging to the **match set** for this cycle. Note that different rules in the match set may advocate different actions.
- The rules in the match set are grouped according to the action they advocate, and a predicted payoff for each action is calculated from the individual rules' predictions.
- Based on these predictions, an action is chosen, and all of the rules that advocated that action are tagged as belonging to the **action set** for that time step.
- The action is posted to the message list. The action consists of instructions to be read by the effectors (which interact with the environment), and (optionally) signals to the left on the “internal” message list.
- Periodically a reward signal is received from the environment. A **credit allocation mechanism** is used to distribute that reward amongst the rules, usually amongst the chain of action sets that led to the reward.
- Periodically the GA is run on the population of rules to generate new rules and delete poorly performing ones.

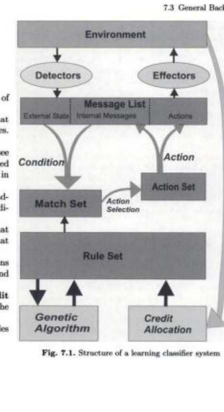


Fig. 7.1. Structure of a learning classifier system

LCS Rules

- Binary with schemata
- E.g. 0#1
  - Satisfied by: (bit1 == 1) AND (bit3 == 0)
  - i.e. bit2 is unspecified, and satisfy both 0 and 1.

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Evolutionary Programming

Pseudo Code

Did I mention you should read in your own time?

Algorithm 3.9.1: Pseudocode for the LCS.

```
Input: EnvironmentDetails
Output: Population
env ← InitializeEnvironment(EnvironmentDetails);
Population ← InitializePopulation();
ActionSet_{t-1} ← ∅;
Input_{t-1} ← ∅;
Reward_{t-1} ← ∅;
while ¬StopCondition() do
    Input_t ← env;
    Matchset ← GenerateMatchSet(Population, Input_t);
    Prediction ← GeneratePrediction(Matchset);
    Action ← SelectionAction(Prediction);
    ActionSet_t ← GenerateActionSet(Action, Matchset);
    Reward_t ← ExecuteAction(Action, env);
    if ActionSet_{t-1} ≠ ∅ then
        Payoff_t ← CalculatePayoff(Reward_{t-1}, Prediction);
        PerformLearning(ActionSet_{t-1}, Payoff_t, Population);
        RunGeneticAlgorithm(ActionSet_{t-1}, Input_{t-1}, Population);
    end
    if LastStepOfTask(env, Action) then
        Payoff_t ← Reward_t;
        PerformLearning(ActionSet_t, Payoff_t, Population);
        RunGeneticAlgorithm(ActionSet_t, Input_t, Population);
        ActionSet_{t-1} ← ∅;
    else
        ActionSet_{t-1} ← ActionSet_t;
        Input_{t-1} ← Input_t;
        Reward_{t-1} ← Reward_t;
    end
end
```

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Evolutionary Programming

ZCS

- LCS can be quite slow at learning / impractical
- Wilson's "Zeroth-Level" Classier System
  - Stripped down LCS
    - No message list

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Evolutionary Programming

ZCS

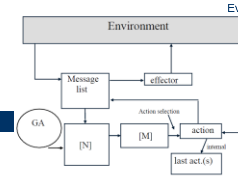


Fig. 1: Schematic of Holland's Learning Classifier System.

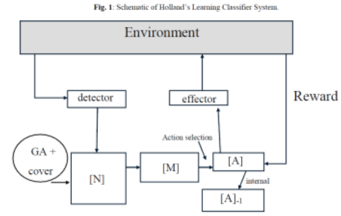
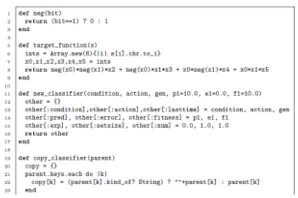


Fig. 2: Schematic of ZCS.

Real Code

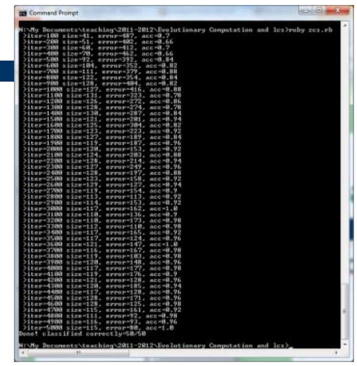
- zcs.rb on BB from Clever Algorithms
- Multiplexer - Static problem



```
def neg(bit)
  return (bit==1) ? 0 : 1
end

def target_function(s)
  ints = Array.new(6){|i| s[i].chr.to_i}
  x0,x1,x2,x3,x4,x5 = ints
  return neg(x0)*neg(x1)*x2 + neg(x0)*x1*x3 + x0*neg(x1)*x4 + x0*x1*x5
end

def new_classifier(condition, action, gen, p1=10.0, e1=0.0, f1=10.0)
  other = {}
  other[:condition],other[:action],other[:lasttime] = condition, action, gen
  other[:pred], other[:error], other[:fitness] = p1, e1, f1
  other[:exp], other[:setsize], other[:num] = 0.0, 1.0, 1.0
  return other
end

def copy_classifier(parent)
  copy = {}
  parent.keys.each do |k|
    copy[k] = (parent[k].kind_of? String) ? ""+parent[k] : parent[k]
  end
```

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Evolutionary Programming

Real Code



A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Evolutionary Programming

Real Code in C

- https://ccrma.stanford.edu/CCRMA/Courses/220b/Lectures/6/Examples/cbn/code/src/zcs.c

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Evolutionary Programming

XCS

- eXtended Learning Classier System
  - http://www.youtube.com/watch?v=jYp_hgwewPc&feature=related

XCS

- Rule fitness not based on payoff received from environment, but on the accuracy of predictions in payoff.
  - Connect LCS with reinforced learning

i) Each rule's error is updated: $\varepsilon_j = \varepsilon_j + \beta(|P - p_j| - \varepsilon_j)$
ii) Rule's predictions are then updated: $p_j = p_j + \beta(P - p_j)$
iii) Each rule's accuracy κ_j is determined: $\kappa = \alpha(\varepsilon_0/\varepsilon)^v$ or $\kappa=1$ where $\varepsilon < \varepsilon_0$
iv) A relative accuracy κ_j' is determined for each rule by dividing its accuracy by the total of the accuracies in the set.
v) The relative accuracy is then used to adjust the classifier's fitness F_j using the moyenne adaptive modifee (MAM) procedure: If the fitness has been adjusted $1/\beta$ times, $F_j = F_j + \beta(\kappa_j' - F_j)$. Otherwise F_j is set to the average of the current and previous values of κ_j'.

A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing
Evolutionary Programming

XCS

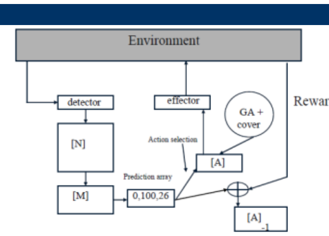


Fig. 3: Schematic of XCS.