Learning Classifier Systems

Andrea Bonarini

Artificial Intelligence and Robotics Lab
Department of Electronics and Information
Politecnico di Milano

E-mail: bonarini@elet.polimi.it
URL:http://www.dei.polimi.it/people/bonarini

Learning Classifier Systems

Genetic algorithms to learn a rule-based model.

Originally, the rules have been used for classification problems, hence the name.

Currently still used for classification, a very general task, which can be interpreted in different ways:

- control
- data mining
- strategy (in games, multi-agent systems, ...)
- ...

The Pittsburgh representation model (I)

A member of the population represents a whole rule base.

...1 1 0 1 1 1 0 0 1 0 0 0 1 0 0 ...

...1 1 0 1 0 1 1 1 0 0 0 0 1 0 0 ...

...0 0 1 0 1 1 1 0 0 1 0 0 0 1 0 ...

...

Rationale:

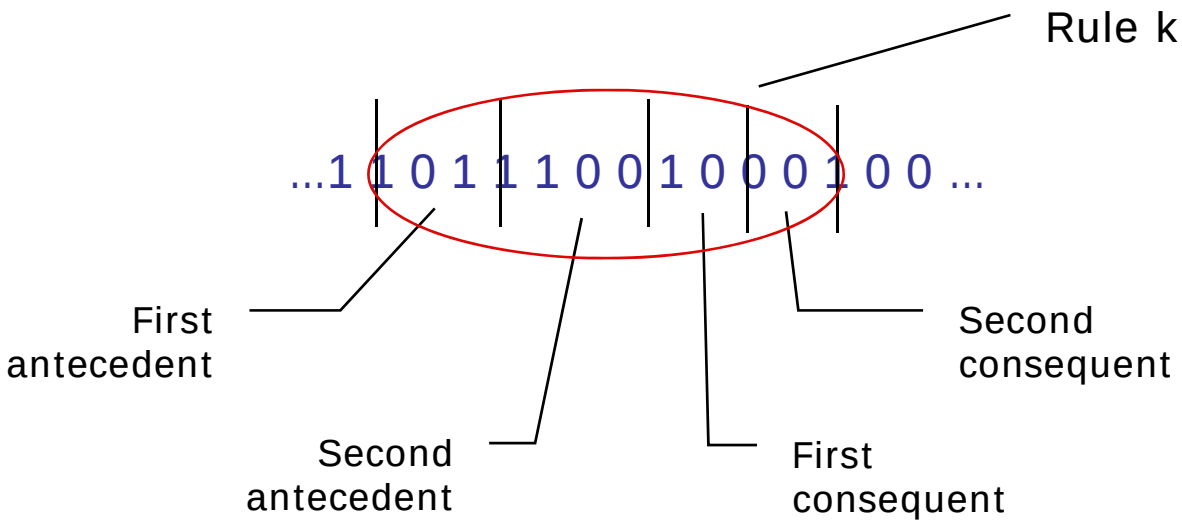
as in GA, a member of the population represents an instance of a possible solution, to be optimized

Potential problems:

Large and fixed number of rules, with fixed structure

Global optimization at each step

The Pittsburgh representation model (II)



E.g.: If (a = 3) And (b = 4) Then (u1 = 2) And (u2 = 0)

If (a = medium) And (b = high)
Then (u1 = medium) And (u2 = zero)

The Pittsburgh learning model

The **fitness** of an individual is computed as the reinforcement obtained in a number of trials by the whole rule base

=> relatively slow learning

The **genetic operators** are redefined to preserve the coding of antecedents and consequents (remember the Building Block assumption), and eventually to focus on rules used in the current trial

The Pittsburgh learning algorithm

Learning algorithm

Repeat

  Fitness evaluation of all the individuals

    ForEach individual:

      1.1 Run the rules of an individual for a given number of steps

      1.2 Evaluate the fitness as the reward accumulated in the trial

  Application of the operators for selection and reproduction

  Application of the operators for recombination (Xover...)

Until the fitness of the best individual is stably over a threshold

The Michigan representation model

A member of the population represents a single rule

1 1 1 0 0 1 0 0 0 1

1 1 0 1 1 0 1 1 0 1

1 1 1 0 1 1 1 1 0 0

...

Rationale:

- Optimization of local models
- The global (optimal) model comes from combination of local (optimal) models

Potential problems:

- Rules with fixed structure
- Global optimization at each step

The learning model

The **fitness** of an individual (a rule) is computed as the (eventually delayed) reinforcement obtained by it in a given number of steps

=> relatively fast local learning

The **genetic operators** are redefined to preserve the coding of antecedents and consequents

The Michigan learning algorithm

Learning algorithm

Repeat

- For a given number of steps:
  - Select a rule and apply it
- Get reward
- Distribute reward to the rules used since the last evaluation
- Apply the operators for selection and reproduction
- Apply the operators for recombination (Xover...)

Until the fitness of the best individual is stably over a threshold

Comparison between Pitts and Mich

Pittsburgh

- Long genes since they code a rulebase
- Fitness evaluation requires to test the whole rulebase
- The number of rules is fixed and large
- The learning model can be easily understood
- Rules in different individuals cannot interact

Michigan

- Short genes since they code only a single rule
- Evaluation of a single rule, also at each step if decided
- The number of individuals is the number of needed rules
- It is not clear how it learns, and what genetics means here
- The single rules compete each other in the same population

Generalization

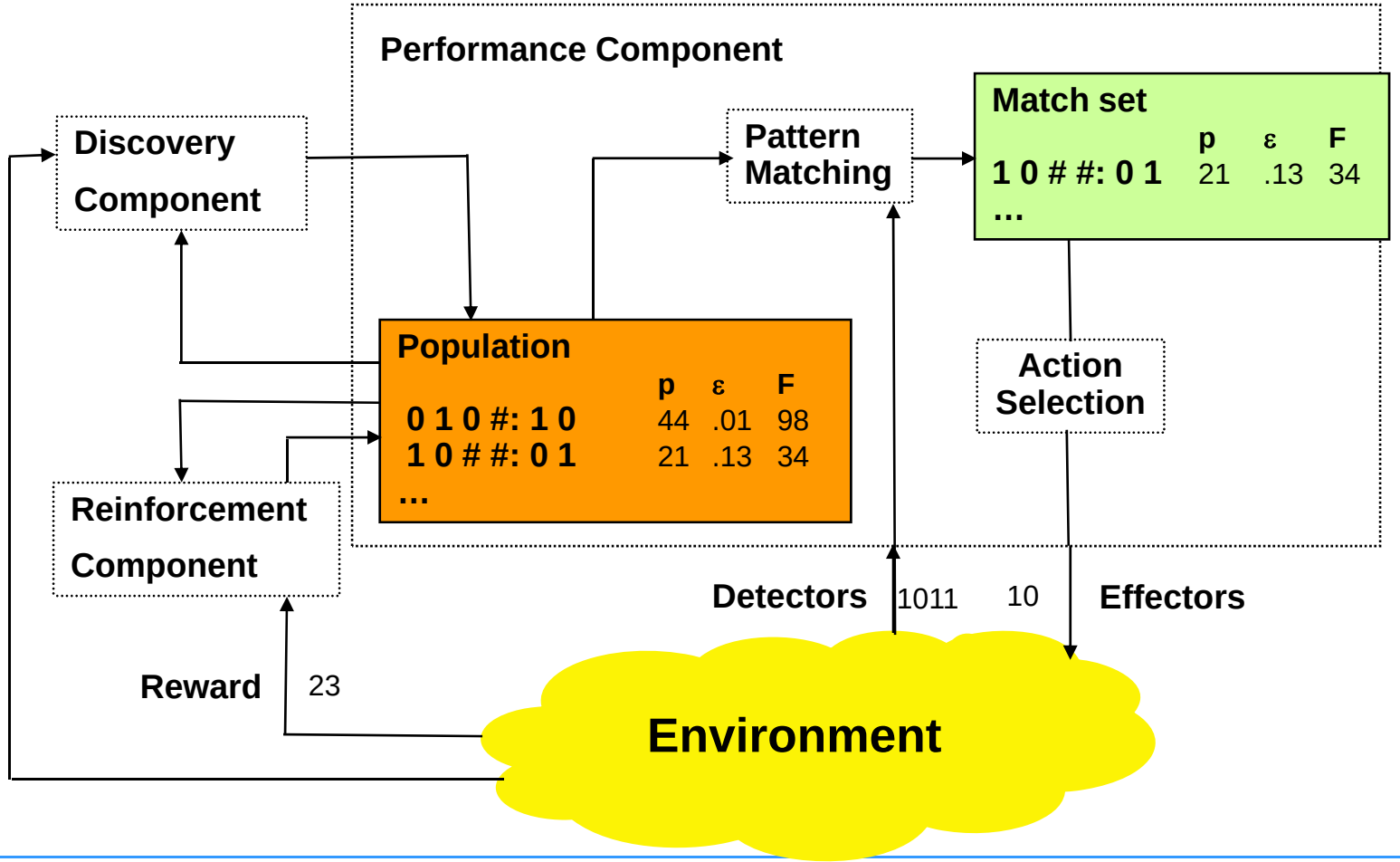
It is possible to extend the number of symbols that can be inserted in an individual, by adding the symbol “don’t care” (#), which stands for an undefined value (in binary coding either 0 or 1).

This introduces a sort of generalization, since a rule containing “don’t care” symbols is more general than one with all the symbols given as 0 or 1

This is used to obtain rule bases with less rules, which put in evidence the relevant aspects

It introduces some relevant aspects in learning due to reinforcement distribution

LCS architecture



XCS a LCS based on accuracy

Parameters

- Prediction (p_j) or Strength (s_j) of the classifier J: estimate of the average reward expected by the use of the classifier
- Prediction error (ε_j): estimation of the validity of p
- Fitness (F_j): parameter used by genetic operators (e.g., to select the mating individuals); it is inversely proportional to ε_j
- Average size of action-sets (as) this classifier belonged to; the smaller as/F is, the less likely it becomes that this classifier is deleted
- Experience (exp) counts how often the classifier belonged to the action set; has some influence on the prediction of other parameters --- namely, if exp is low default parameters are used when predicting the other parameter (especially, for ε, F and as)

Action selection

Prediction array

For each action of the matching classifiers the expected value is computed as:

$$p(a) = \frac{\sum_{Action(C_j)=a} p_j F_j}{\sum_{Action(C_j)=a} F_j}$$

Then the action is selected, by using p(a) as a seed for a *roulette wheel* selection

Credit assignment

The parameters of each classifier that has contributed to obtain reward are updated according to Q-learning-like formulas

Given $P = \max_{a \in A} p(a)$

$$\varepsilon_{j=}\varepsilon_j + \beta \, [\, |P - p_j| - \varepsilon_j]$$

The accuracy k_j of a classifier j is updated by:

$$k_j = \exp[(\ln \alpha)(\varepsilon_j - \varepsilon_0)/\varepsilon_0)] \text{ for } j > 0, \text{ otherwise } 1$$

The relative accuracy k_j' is obtained by dividing k_j by the sum of the accuracy values of the matching set (a *niche*).

Then, F is updated, as the average of F if exp is low, otherwise as:

$$F_j = F_j + \beta \, [\, k_j' - F_j]$$

Finally p

$$p_j = p_j + \beta \, [r + \gamma \, P - p_j \,]$$

Discovery component

At the beginning the population can be randomly generated, initiated with guesses, or even empty.

If no classifier matches the input, a cover *detection operator* is applied: a rule matching the input with a given number of # distributed in it, and a random output is generated, inserted in the population, and applied.

At given time points, the *genetic operators* are applied, a pair of classifiers are selected from the action set, mated, and the generated offsprings are included in the new population. The values of the parameters of the offsprings are the average of those of the parents.

When the population reaches the maximum dimension and new offsprings are to be introduced, some classifiers are deled by selecting in probability the worst fitting, or the ones that participate in the largest match sets it participates to.