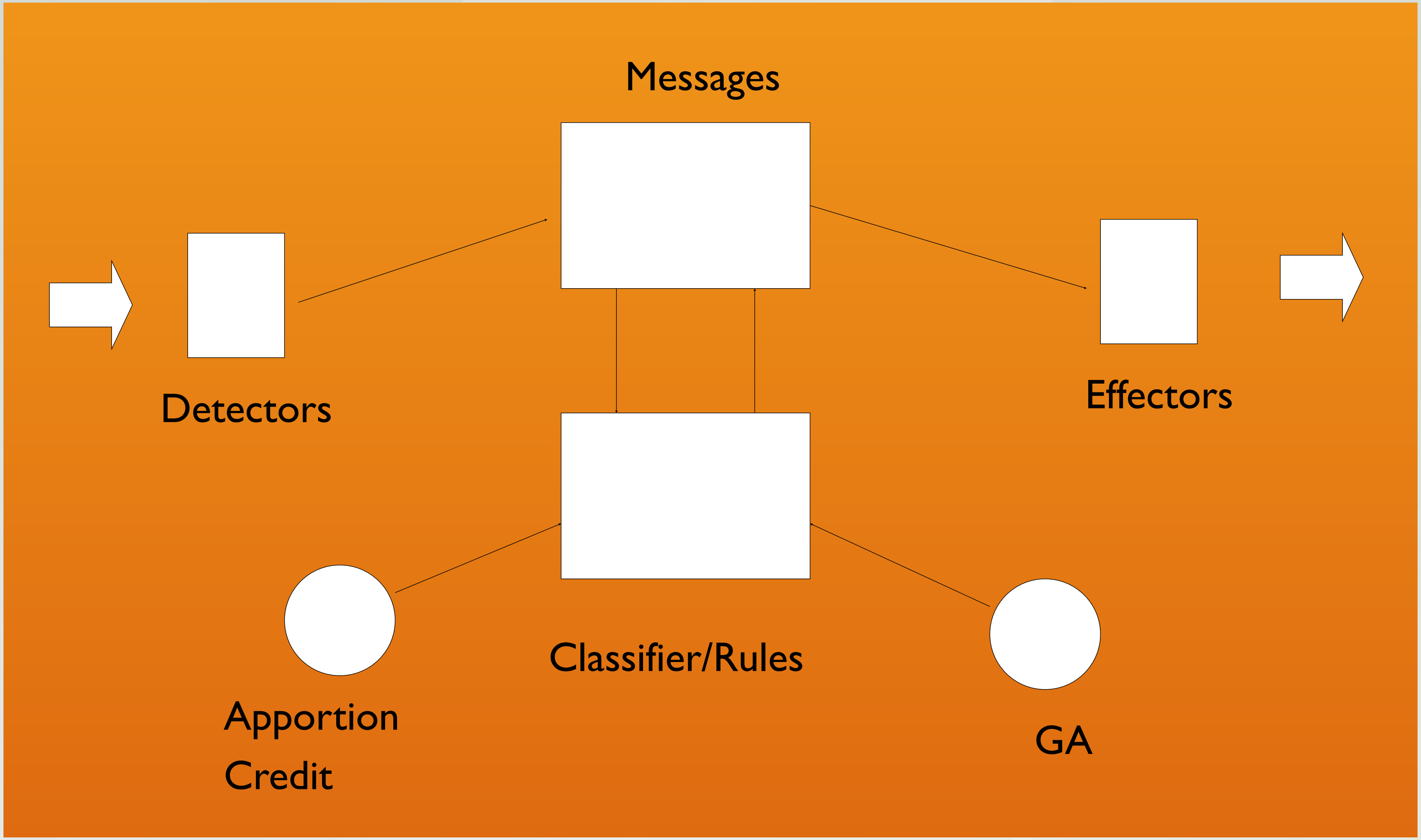
CSE/ECE 848
Introduction to
Evolutionary Computation

Module 3 - Lecture 15 - Part 5
Learning Classifier Systems

Wolfgang Banzhaf, CSE
John R. Koza Chair in Genetic Programming

A Classifier

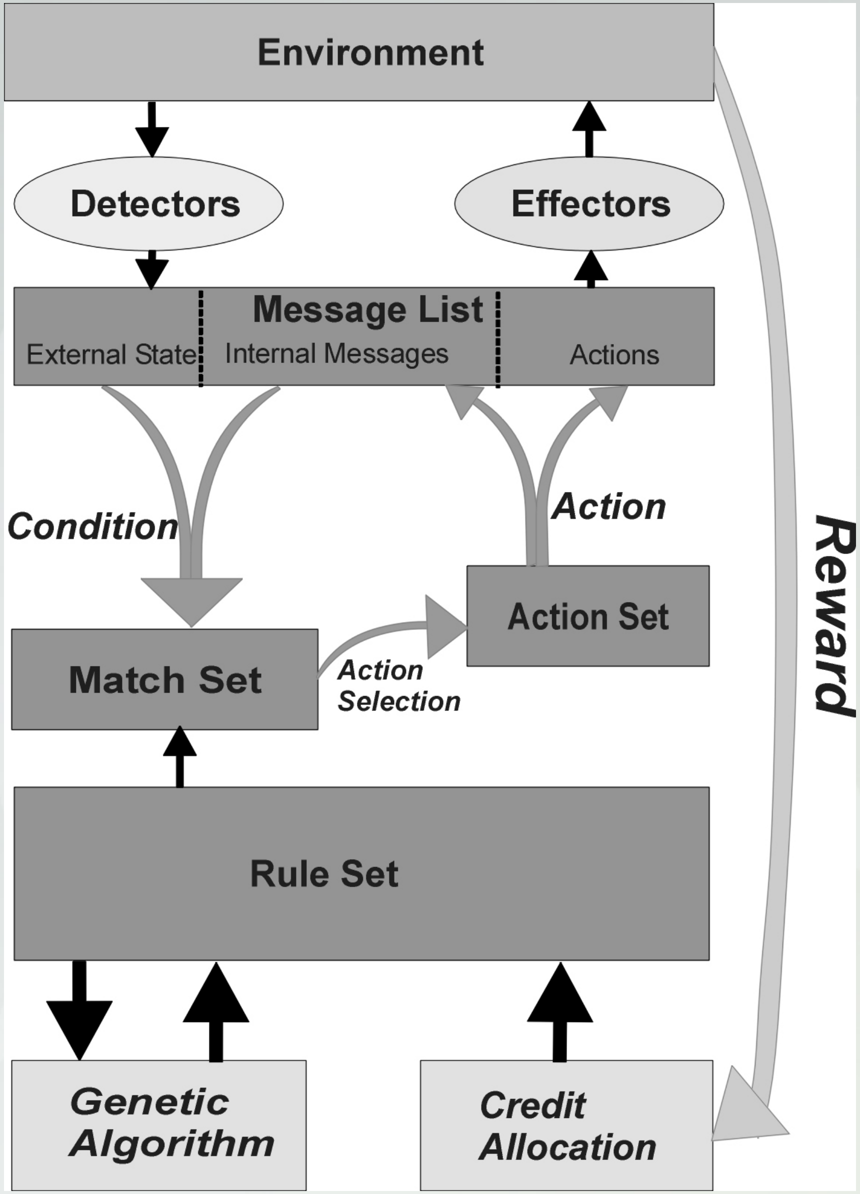


Messages and Rules

Like GAs in general, the alphabet for Classifiers is binary:

- <condition> ::= {0,1,#}*
- <message> ::= {0, 1}*
- <classifier> ::= <condition>:<message>

- The "#" is the wildcard we've seen before in schema discussions. It matches any character.


Environment
Detectors
Effectors
Message List
External State
Internal Messages
Actions
Condition
Action
Reward
Action Set
Match Set
Action
Selection
Rule Set
Genetic
Algorithm
Credit
Allocation

Basic Operation

- Detector posts a message to the list
- Message list may also contain internal state information posted by other rules
- All rules are examined and those whose conditions match against the message list are the match set M
- Rules in the match set are grouped according to their actions

Basic Operation II

- Based on various values, an action is selected and all rules with that action from the match set are the action set A.
- An action is posted to the message list. If action messages match an effector, external action is taken.

Basic Operation III

- A reward is occasionally distributed based on rule performance
- Learning must be added if the system is to become better over time

Rating Rules

- The problem with allowing all rules to match is controlling which ones get to post to the message list.
- We would like to allow those rules that are most "profitable" in the list to post over other rules.
- But determining profitable rules is hard. They work as chains, not as individuals.

Reinforcement Learning

- Turns out this is exactly the problem that reinforcement learning addresses. You cannot rate the goodness of every action as it is performed.

- In “real life”, you do a number of actions that look “right”, then when feedback finally comes you update the reward that the sequence "deserves"

Reinforcement Learning II

Not our topic here, but useful to know a little about RL.

- Focus is usually on learning a *policy*, a function that describes to an agent what to do from a particular state
- This policy is *learned* by repetitive action and feedback on those actions
- Can have probabilities of success on any action
- Is a very realistic model for real agents in the world

Reinforcement Learning III

- Is therefore focused on exploration vs exploitation (sounds familiar?)
- Needs iteration to arrive at a final answer
- Can be used to search for a "good" answer that may or may not be optimal (but might be optimal given certain circumstances)

Another Analogy: Economies

- We will use the analogy of an economy to try and control rule strength and their abilities to post actions.
- Rules will have some associated "worth", and each rule must "bid" for the right to post its message based on its worth.
- Rules that are "profitable" receive a downstream reward for their posting, renewing their worth.

The Bucket Bridgade

- Strength is modified by:

$$S_i(t+1) = S_i(t) - P_i(t) - T_i(t) + R_i(t)$$

- Where P is payment, T is tax and R is receipts.
- Tax is necessary to have a rule eventually run out of resources if it doesn't perform. Receipts is the downstream payoff for participation.

Bidding

- Bids are done in fixed proportion to their strength:

$$B_i = C_{bid} S_i$$

- To maintain “diversity”, we introduce some randomness into the bid based on some distribution:

$$EB_i = B_i + N(\sigma_{bid})$$

On Strength Calculation

- Many taxing methods, easiest is some constant:

$$T_i = C_{tax} * S_i$$

- Final equation looks like:

$$S_i(t+1) = S_i(t) - C_{bid} S_i(t) - C_{tax} S_i(t) + R_i(t)$$

- Receipts are calculated by payments made and reward, divided among the participating rules.

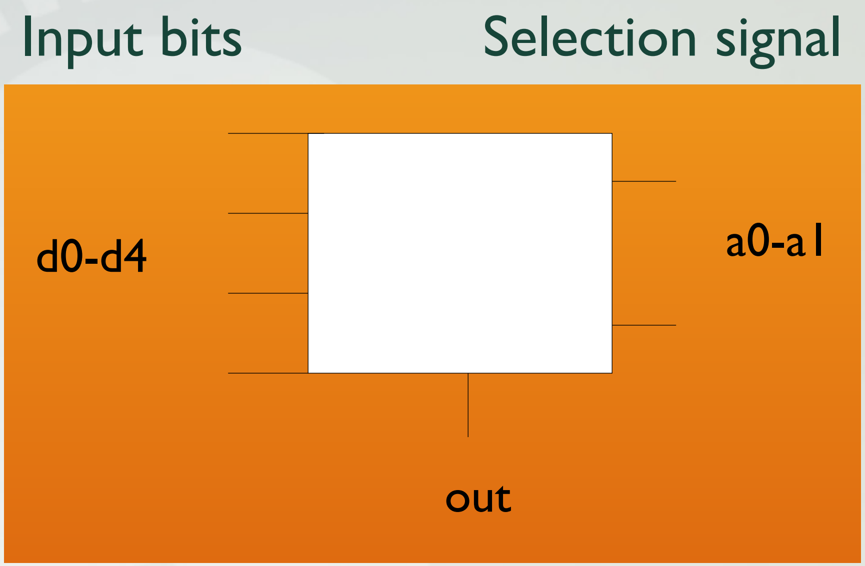
Learning, via GA

The learning approach is similar to what we have seen previously with the following exceptions:

- We don’t replace the entire population each generation, to preserve some rules
- We must let the Classifier run a bit to develop rule strengths, so the GA only runs every x generations.

Simple Example: The 6 Multiplexer

- Perfect rules:
- ###000:0
- ##0#01:0
- #0##10:0
- 0###11:0
- ###100:1
- ##1#01:1
- #1##10:1
- 1###11:1



Default Hierarchies

- Order the rules, from specific to default, to reduce the number of rules!

- ###000:0
- ##0#01:0
- #0##10:0
- 0###11:0
- ######:1 (default condition, check last)

Default Hierarchies II

Some advantages of default hierarchies:

- Parsimonious rules: requires fewer rules to get the job done
- Enlarges the solution space: multiple rulesets can work together
- Knowledge Acquisition: natural organization of knowledge

How to create Hierarchies?

- Create hierarchies by making the specificity of the rule (the number of fixed positions in the condition) a part of the bidding process.

$$B_i = C_{bid} * (bid1 + Sp * bid2) * S_i$$

- This allows specific rules to outbid nonspecific rules.