Artificial Intelligence Programming
Classifier Systems

Chris Brooks

Department of Computer Science
University of San Francisco
6 Genetic Algorithms work very well for optimization problems
   \[ f(x_1, x_2, x_3, \ldots, x_n) = R \]
6 What about less well-defined problems?
6 Can we use these ideas to build a vacuum-cleaner agent?
6 (yes, but we’ll need to be a bit more careful)
6 Classification is the process of assigning an input to one of several classes.
   \[ \text{We’ll see much more of this later on.} \]
Production systems are a common AI technique

if-then, or condition-action rules

if [1,1] has not been vacuumed and is adjacent, then move to [1,1]

A Production system will consist of a large body of rules.

- Some rules may cause other rules to fire.

In traditional systems, the problem is what to do when more than one rule is matched.

(We’ll return to this in a few weeks.)
Genetic Algorithms potentially give us a way to evolve and select rules.

- Pick the rule with the highest fitness.

Encode rules as a bitstring, with 0,1,*

- condition bits : action bits

Issues:
- How to assign fitness to a rule?
- How to maintain high online performance?
A Classifier system has three components:

- A rule and message system
- A credit assignment system
- A GA for generating new rules
4-4: Rule and message system

- The agent’s sensors receive information, which is encoded as a bitstring.
  - This information is a message from the environment.

- This message activates classifiers (rules) with matching conditions.

- These classifiers post their messages to a message list.
  - These messages may activate other classifiers, or the agent’s effectors.
Assume our system has the following classifiers:

1) 01## : 0000
2) 00#0 : 1100
3) 11## : 1000
4) ##00 : 0001

Message 0111 arrives from the environment.

- Rule 1 fires, placing 0000 in the message list.
- Rules 2 and 4 fire, placing 1100 and 0001 in the message list.
- Rule 3 fires, placing 1000 in the message list.
- This matches rule 4, whose message is in the message list.

There are several messages now in the message list - which is sent to the effectors?
The bucket brigade algorithm allows rules to “bid” on firing based on past performance.

When a rule is matched, it participates in an “auction.”

Each rule has a strength, based on past performance.
- A rule bids a constant proportion of its strength.

Highest bidding rules win out.

This bid is sent to the classifier(s) that activated it.
Initially (t = 0): M = 0111

1) 01## : 0000  S = 200  B = 20
2) 00#0 : 1100  S = 200
3) 11## : 1000  S = 200
4) ##00 : 0001  S = 200

Environment  S = 0
t = 1

1) 01## : 0000  S = 180  0000
2) 00#0 : 1100  S = 200  B = 20
3) 11## : 1000  S = 200
4) ##00 : 0001  S = 200  B = 20

Environment  S = 20
4-9: Example

6 \( t = 2 \)

1) 01## : 0000 \( S = 220 \) 1100
2) 00#0 : 1100 \( S = 180 \) 0001
3) 11## : 1000 \( S = 200 \) \( B = 20 \)
4) ##00 : 0001 \( S = 180 \) \( B = 18 \)

Environment \( S = 20 \)
6  t = 3

1) 01## : 0000   S = 220   1000
2) 00#0 : 1100   S = 218   0001
3) 11## : 1000   S = 180
4) ##00 : 0001   S = 162   B = 16

Environment   S = 20
t = 4

1) 01## : 0000  S = 220  0001
2) 00#0 : 1100  S = 208
3) 11## : 1000  S = 196
4) ##00 : 0001  S = 156

Environment  S = 20
t = 5 - payment comes into system, assigned to last active classifier.

1) 01## : 0000 S = 220
2) 00#0 : 1100 S = 208
3) 11## : 1000 S = 196
4) ##00 : 0001 S = 206

Environment S = 20
Bucket brigade provides a way of selecting rules and assigning credit.

How to get new rules?

Our basic GA uses a nonoverlapping population model.
- All the population at time $t$ is replaced at time $t + 1$

This works well for optimization, but not so much for learning.

Instead, we use elitism to retain some rules.

Use roulette selection to retain rules.

Also, evolve more slowly - every $T$ iterations rather than 1.
Evolutionary techniques have also been applied to the construction of programs.

Idea: A program can be represented as an *S-expression*

```
(+ 3 (* 4 x))
```

We can draw this as a tree.

We then perform evolution on a population of programs.
Fitness is the performance of a program on a given task.

Crossover swaps subtrees.

Mutation replaces operators.

Used to evolve:
- Robot soccer programs
- Analog control circuits
- Filters
- Complex circuits

Challenges:
- Vast (infinite) space of programs
- Lack of higher-level program structures