Part II — Evolutionary Learning

Jonathan Shapiro

Department of Computer Science, University of Manchester

June 18, 1999

A simple example — GABIL

Reference: De Jong

Overview: Learns a boolean concept by searching for a disjunctive set of propositional rules.

Representation of a rule: A substring for each variable in a rule, with a bit for every value which that variable can take plus a string for the concept value.

Example: if x takes values X_1, X_2, X_3 and y takes values Y_1, Y_2, then the string

$$110\ 01\ 1$$

is used to denote the rule

$$\text{IF } (x = X_1 \vee x = X_2) \wedge (y = Y) \text{ THEN } c = \text{TRUE},$$

where c is the concept.

Set of Rules: a long string of rule substrings concatenated together

Objective function: (number of examples correctly classified)^2

Crossover operator: allowed for variable-length rule sets.

- Two-point crossover used.
- Crossover points in second parent can be in any rule, but in analogous place in the rule as in the rule of first parent.

	rule 1			rule 2			rule 3		
	x	y	c	x	y	c	x	y	c
parent 1:	00|1	11	1	111	0|1	0	110	10	1

	rule 1			rule 2			rule 3		
	x	y	c	x	y	c	x	y	c
parent 2:	10|1	1|	1	0	001	01	0		

	rule 1			rule 2			rule 3		
	x	y	c	x	y	c	x	y	c
offspring 1:	001	11	0	110	10	1			
offspring 2:	101	11	1	111	01	1	001	01	0

Note that rules with more 1 are more general, e.g. all 1's matches any value.

Mutation: Biased towards introducing 1's.

- Operator which changes 0 to 1
- Operator which changes all bits associated with a variable within a rule to 1 (rendering that variable irrelevant in that rule).
- Ordinary unbiased mutation

Results: Compared with AQ14, ID5R, C4.5; found to give comparable results with unbiased mutation. Improved when biased mutation operators used.

Is the population the “answer” or is the best member of the population the “answer”?

Holland’s original idea was to evolve rule-sets, where the rule-set would be the population.

- The “Michigan school”

DeJong evolved populations of rule-sets (like most work nowadays)

- The “Pittsburgh school”

Next we see some examples in which the entire population is used as the answer.

Time-series prediction

Reference: Packard, 1990, Meyer and Packard, 1992.

Task: Given a value of some quantity x now, and some number of time-steps ago, predict the value of x a time T in the future. I.e. given the *lag vector*

$$(x_t, x_{t-\tau}, \ldots, x_{t-n\tau})$$

predict x_{t+T}.

Modified Task: Recognise *if* prediction is possible; only then do prediction. Look for the "windows of predictability".

Purpose: Find a population of rules which are predictive. For situations where no rule in the population is satisfied, prediction may not be possible.

Rule Set: Conjunctions of ranges (variable number of terms), e.g.

$$\{(3.2 \leq x_6 \leq 5.5) \wedge (0.2 \leq x_2 \leq 0.3) \wedge (3.4 \leq x_{10} \leq 9.8)\}$$

Objective Function:

$$-\log\left(\frac{\sigma}{\sigma_0}\right) - \frac{\alpha}{N_c}$$

- σ is the standard deviation of the value T steps in the future for all data points which satisfy this rule;
- σ_0 the standard deviation T steps in the future for all data
- N_c is the number of points which satisfy the rule;
- α is a constant.

Selection: Rank the population by fitness; throw out and replace the low fitness ones.

Crossover: Uniform crossover, taking one range condition as a bit.

Mutation: Four types used:

- Add a new condition
- Delete a condition
- Broaden or shrink a range
- Shift a range up or down.

Results: Tested on the Mackey-Glass equation, a chaotic, non-local dynamical system. Found rules with small variances.

Learning Classifier Systems

Reference: Holland, 1986

Holland proposed a learning system based on his hypotheses about the efficiency of genetic algorithms and natural adaptive systems.

Rule-set: a production rule system.

condition	message	action	strength
101#####	011010	110	[56]

System: A set of rules, receiving inputs from a message list and an input system; producing new messages and outputs to an output system.

Task: Reinforcement learning, system maps inputs to actions. Occasionally receives a reward.

Functioning:

1. Inputs put a message on the message list
2. Some rules match a message on the message list.
3. Those rules compete to respond to the message, based on their strength (and sometimes on the specificity of the rule).
4. The winner must pay (lose some strength) the sender of the message; it puts its message on the message list and its action is performed.
5. When reinforcement is received; it is given to the classifier posting the action. (Called a bucket-brigade algorithm.)

Adaptation: The set of rules is a population of a genetic algorithm. Crossover and mutation are used to generate new rules. Steady-state selection, with niching or similar to enforce different rules is used to propagate the effective rules. The strength of the rule is selected for.

Default Hierarchies: Rules which are specific (few #'s) are favoured in the conflict resolution and credit assignment schemes. Learning classifiers can adaptively create default hierarchies.

Default rule: A general rule (many #'s) which works in most but not all cases.

Exception rule: A specific rule which is satisfied by a subset of the messages matched by the default rule, and whose action corrects that of the default rule.

Using the population as a committee

Reference: Yao and Liu, 1996

A third way of using the entire population as the solution is to combine the outputs as a committee.

- Applied to training neural networks; but could be used for any learning system.
- Each member of the population represents a potential entire solution (as in the GABIL system).
- Implicit fitness sharing used to encourage different population members to perform correctly on different subsets of the example set.

- After GA is finished, entire population is used to produce output:
  - Majority voting
  - Voting weighted by fitness rank
  - Other methods tried

  Combined system was significantly better than the best member of the population on a majority of the real-world problems considered, but not on all.

Potentially, an inferior method of creating a committee machine due to the lack of independence of population members; but superior to throwing out all of the learning systems in the population but one.

Genetic Programming

Reference: Koza, 1992, 1994;

Uses genetic algorithms to generate computer programs represented as parse trees.

Representation: Parse trees of LISP expressions. Requires:

Terminals.

Functions which take other functions or terminals as input.

Example: Representing the absolute value function.

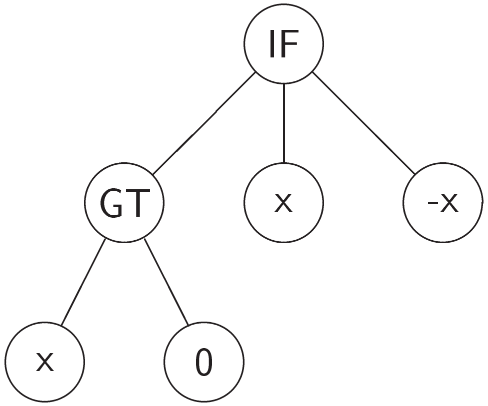
Terminal set: real variables x; real constants; boolean constants.

Function set: IF$(\cdot, \cdot, cdot)$ evaluates the second (third) argument if the first evaluates to TRUE (false). GT$(\cdot, \cdot)$ returns true if the first argument is greater than the second.

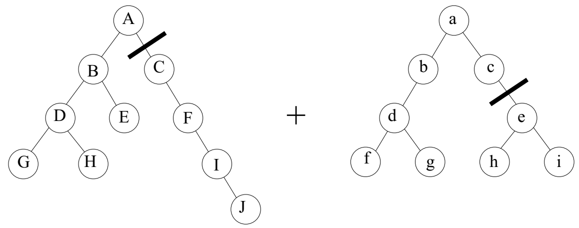
Representation of program as a functional expression:

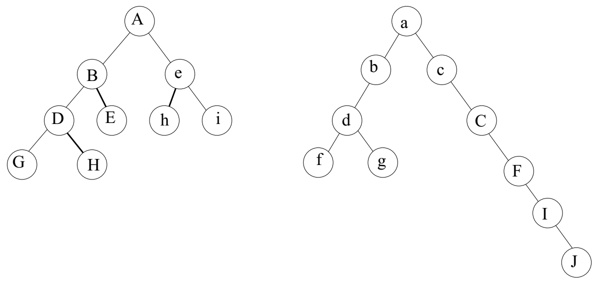
```
(IF (GT (x) (0))
    (x)
    (-x))
```

An expression represented as a tree:



Crossover: One randomly chosen subtree from one parent is replaced by that of another.





Mutation: Take a random subtree and replace it with a randomly generated one (often not used). Generates variable sized structures.

Selection: Fitness proportional selection with 10% elitism.

Example: Simulated Ant

Scenario: A simulated ant can detect food directly in front, can turn left or right, and can move forward. The world is a grid with a trail of food. There are some gaps in the trail.

Task: Collect as much food as possible from the trail. Since there are gaps, the ant must have exploration behaviour for when no food is detected, to find the trail.

Terminals: MOVE, RIGHT, LEFT.

Functions: IF-FOOD-AHEAD$(\cdot,\cdot)$ evaluates the first argument if food is ahead; the second if it is not. PROG2$(\cdot,\cdot)$ evaluates its first argument and then the second. PROG3$(\cdot,\cdot,\cdot)$ evaluates its three arguments in turn.

Examples: This moves towards food if detected, else turns left.

```
(IF-FOOD-AHEAD (MOVE) (LEFT))
```

Fitness: The number of pieces of food found by the ant in a given number of time-steps. Program is iterated until the number of time-steps is exceeded.

Perfect solution: (for the particular trail) found after 21 generations with a population size of 500.

```
(IF-FOOD-AHEAD (MOVE)
               (PROGN3 (LEFT)
                       (PROGN2 (IF-FOOD-AHEAD (MOVE)
                                              (RIGHT))
                               (PROGN2 (RIGHT)
                                       (PROGN2 (LEFT)
                                               (RIGHT))))
                       (PROGN2 (IF-FOOD-AHEAD (MOVE)
                                              (LEFT))
                                (MOVE))))
```

Evolving a cellular automata to perform a task.

References: Mitchell, Crutchfield, Das, 1996

Cellular automata computer:

- One dimensional line of cells, each taking 0 or 1 as its state.
- Each cell takes the state of r of its neighbours on each side and its own state and produces a new state according to a rule.
- All cells use the same rule, and are updated in parallel.
- System input: initial condition of the CA.
- System output: final condition of the CA.
- Programming task: find the rule which implements a given input-output relationship.

Example: Density Classification (Gacs, Krudyamov, Levins, 1978).

- Output: all 1's if more than half the inputs were 1's
- Output: all 0's otherwise.

Genetic algorithm used to search the 2^7 $(r = 3)$ bit rule table.

Selection: Take top 20% (elitism); modify bottom 80%.

Fitness: Number correct on 100 initial conditions with uniformly distributed densities.

Results:

- "Epochs of innovation" — dynamics exhibits punctuated equilibria
- "Information carrying particles". Interpreted the solutions as developing mechanisms to transmit information from one side of the system to the other. This was through domains of particular spatio-temporal patterns, the interfaces of which were interpreted as information carrying particles.

Genetic Algorithms and Neural Networks

References: Schaffer et. al. 1992; Yao, 1995

Genetic algorithms have been used for three general tasks in neural network learning:

1. Use a genetic algorithm to find the weights of a fixed architecture neural network.
2. Use a genetic algorithm to find the structure or architecture of the neural network (using perhaps another algorithm to learn the weights).
3. Use a genetic algorithm to evolve the learning rule for a neural network (Chalmers, 1990).

Genetic Algorithms to Train Fixed-Architecture Neural Networks

For supervised learning, this appears to be a questionable enterprise.

1. If gradient information is available, it is usual best to use it.
2. Genetic algorithms turn all problems into reinforcement learning problems (performance is defined by one number). Reinforcement learning algorithms are generally slower than supervised learning ones.

The argument in favour of GAs is that they are global search algorithms.

A number of researchers have studied this application, but there is little evidence that it is better than the standard algorithms (except for discrete-valued weights).

Genetic algorithms to search Neural Network Architectures

An evolutionary approach can be used to evolve the architecture of the network. Example EPNet (Yao, 1995).

Training the weights: done with a adaptive gradient descent/simulated annealing hybrid. Run for a fixed number of steps each generation.

Mutations: Grow and prune the network.Try in order

- delete a node
- delete a node
- add a node
- add a connection

until something works. (Biased towards pruning.)

Training sets: A training set used to train the weights; a validation set is used to evaluated fitness for architecture construction; a second validation set is used to select a member of the population to use as the final classifier.

Adding crossover

To include crossover, the architecture must be represented as strings. Two methods:

directed encoding: structure of the network is represented directly in the string. For example, the connectivity matrix could be represented as a string by rasterization (Miller et. al. 1989).

indirect encoding: The string encodes some process which generates the network.

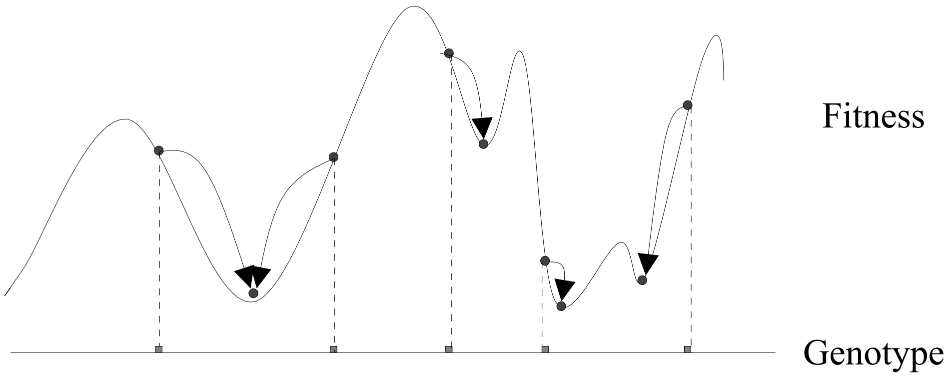
- **grammatical encoding** the strings represent a grammar (Kitano, 1990) which generates the network.
- **developmental encoding:** here the strings represent a developmental process which generates the network architecture (Miglino et. al. 1996.)

Combining local search with evolutionary search

References: Belew and Mitchell (eds) "Adaptive Individuals in Evolving Populations", 1996; Evolutionary Computation, Vol 4, #3, 1996, special issue on the Baldwin Effect, Belew 1989, Ackley and Littman, 1991.

It is often useful to combine genetic algorithms with local algorithms. These are sometimes called memetic algorithms.

- Evolutionary search moves members of the population near local optima
- Local search takes them the rest of the way.



Sometimes called “memetic” algorithms.

Darwinian or Lamarkian

Which strings should go into the next population?

- The strings modified by local search? (Lamarkian)
- The strings unmodified by local search? (Darwinian)

The case for Lamark

Reference: Hart and Belew, 1996. Looked a simple problem

Objective function:

$$f_\sigma = \frac{\sigma}{4000}\sum_{i=1}^{n} x_i^2 + \left(1 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)\right).$$

Local search: Conjugate gradient and Solis-Wets.

GA-Local search hybrids: compared the Lamarkian and Darwinian approaches.

Results: Lamarkian approaches were significantly better than the other approaches in most of the trials. Darwinian was better when the local searches were infrequent.

Interpretation: Initially, the population is distributed across the state space. With a GA alone, it would correlate around one peak. With frequent local searches, the population remains more diverse; Lamarkianism enhances this.

The case for Darwin

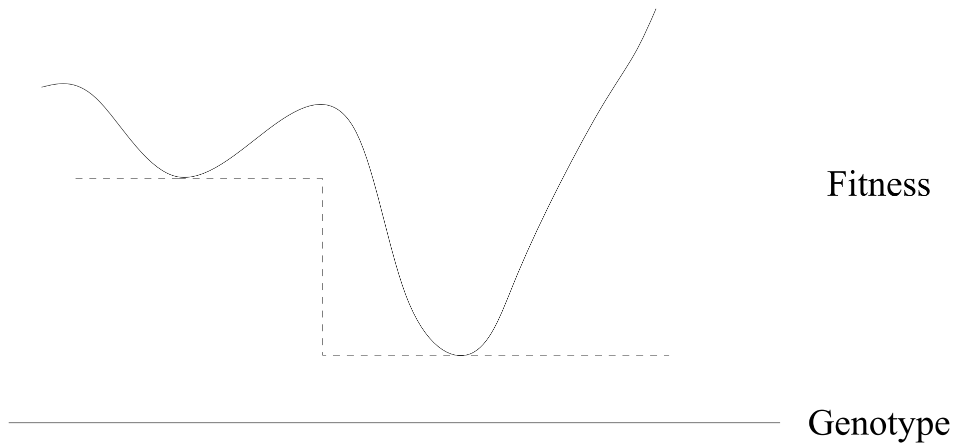
References: Wales and Doye, 1999

Observation: Darwinian combining of evolutionary and local search is equivalent to modification of the objective function.

Interpretation: All strings within a basin of attraction of an optima have the same effective objective function. Thus, the population remains more diverse.

Application: Finding clustering structure of atoms, genetic algorithm/local search hybrids are among the most successful algorithms. Wales and Doye introduce the *basin-hopping* algorithm which implements this objective function modification in simulated annealing.

Results: Are comparable to the genetic algorithm.


Fitness
Genotype

The Baldwin Effect

J. M. Baldwin, 1896

Interaction of learning and evolution could lead to indirect inheritance of acquired characteristics.

<u>Two steps</u>

1. Plasticity (e.g. learning) can increase fitness of partial mutation (otherwise useless).

2. Genetic assimilation — adaptation becomes genetically encoded and innate.

   - plasticity has costs.

Seemingly Lamarkian, but no transfer of information from soma to germ cell.

Hinton - Nowlan Model (1987)

Genotype: string of 1's, 0's, ?'s ($L = 20$).

$$101??0?1\cdots$$

Phenotype: Perceptron with 0-1 weights.

Ideal phenotype: Perceptron with all weights equal to 1.

Plasticity: Randomly assign 1 or 0 to each ? weight. 1000 trials.

Fitness:

$$20 - \frac{19}{1000}\text{min}\left[\#\text{ trials to find ideal}, 1000\right]$$

Learning has a cost.

Without learning: A needle in a haystack problem; solution is never found.

With learning: Search problem is smoother. Ideal solution found after about 10 generations ($N = 1000$). Proportion of ?'s in genotype decays to zero; proportion of 1's goes to 1.

Conclusions:

- Learning guides evolution.
- With plasticity, evolution can work even where only a perfect solution imparts fitness advantage (J. M. Smith).