Chapter 9

GENETIC ALGORITHMS

1. Motivation:

- Genetic Algorithms (GAs) provide a learning method motivated by an analogy to biological evolution.
- Rather than search from general-to-specific hypotheses, or from simple-to-complex, GAs generate successor hypotheses by repeatedly mutating and recombining parts of the best currently known hypotheses.
- At each step, a collection of hypotheses called the current population is updated by replacing some fraction of the population by offspring of the most fit current hypotheses.
- The process forms a generate-and-test beam-search of hypotheses, in which variants of the best current hypotheses are most likely to be considered next.

2. Genetic Algorithms

The problem addressed by GAs is to search a space of candidate hypotheses to identify the best hypothesis.

In GAs, the "best hypothesis" is defined as the one that optimizes a predefined numerical measure for the problem at hand, called the hypothesis fitness.

Exs: Learning task	Fitness
① The problem of approximating an unknown function given training examples of its input and output	The accuracy of the hypothesis over this training data.
② A strategy for playing chess	The no. of games won by the individual.

Structure of Genetic Algorithms:

Although different implementations of genetic algorithms vary in their details, they typically share the following structure:

- The algorithm operates by iteratively updating a pool of hypotheses, called the population.
- On each iteration, all members of the population are evaluated according to the fitness function.
- A new population is then generated by probabilistically selecting the most fit individuals from the current population.
- Some of these selected individuals are carried forward into the next generation population intact.
- Others are used as the basis for creating new offspring individuals by applying genetic operations such as crossover and mutation.

A prototypical genetic algorithm is given below:

GA(Fitness, Fitness_threshold, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.

Fitness_threshold: It specifies the termination criterion.

p: The no. of hypotheses to be included in the population.

r: The fraction of the population to be replaced by crossover at each step.

m: The mutation rate.

- Initialize population: $P \leftarrow$ Generate p hypotheses at random.
- Evaluate: For each h in P, compute Fitness(h).
- While $[\max_h \text{Fitness}(h)] <$ Fitness_threshold do

Create a new generation, P_S:

1. Select: Probabilistically select $(1-r) \cdot p$ members of P to add to P_S. The probability $Pr(h_i)$ of selecting hypothesis h_i from P is given by

$$Pr(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{j=1}^{p} \text{Fitness}(h_j)}$$

2. Crossover: Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P, according to $Pr(h_i)$ given above.

For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator.

Add all offspring to P_S.

3. Mutate : Choose m percent of the members of P_S with uniform probability. For each, invert one randomly selected bit in its representation.

4. Update : $P \leftarrow P_S$.

5. Evaluate : for each h in P, compute Fitness(h).

- Return the hypothesis from P that has the highest fitness.

2.1 Representing hypotheses

Hypotheses in GAs are represented by bit strings, so that they can be easily manipulated by genetic operators such as crossover and mutation.

Ex: Representing sets of if-then rules

Attribute : Outlook
Values : Sunny, Overcast, Rain

(i) Outlook = Overcast — 010 (Overcast is 2nd value, so 2nd bit is 1)

(ii) (Outlook = Overcast v Rain) — 011

(iii) (Outlook = Overcast v Rain) ∧ (Wind = Strong)

Outlook Wind
011 10
(bit string of length five)

(iv) Rule postconditions (such as PlayTennis = yes) can be represented in a similar fashion.

If (Wind = Strong) Then (PlayTennis = yes)
is represented as

Outlook	Wind	PlayTennis
111	10	10

(111 for Outlook indicates that any value for Outlook can be taken [don't care constraint].)

(for PlayTennis: yes / no — 2nd value; 1st value)

2.2 Genetic operators

The generation of successors in a GA is determined by a set of operators that recombine and mutate selected members of the current population.

The two most common operators are:
crossover and mutation.

Crossover:

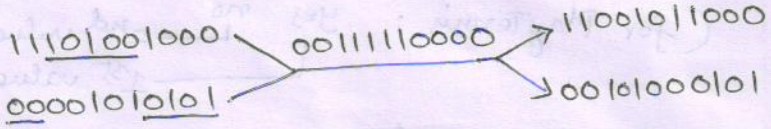
It produces two new offspring from two parent strings, by copying selected bits from each parent. The bit at position i in each offspring is copied from the bit at position i in one of the two parents.

Crossover mask:

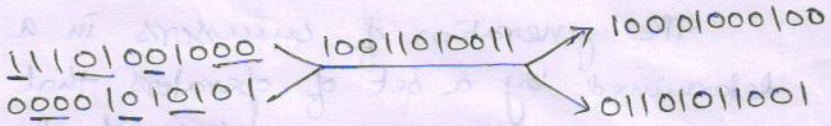
The choice of which parent contributes the bit for position i is determined by an additional string called the "crossover mask".

Operator	Initial strings	Crossover mask	Offspring
Single-point crossover:	11101001000 00001010101	11111000000	11101010101 00001001000

Two-point crossover :



Uniform crossover :



Point mutation : 11101001000 ⟶ 11101011000

Explanation :

Single-point Crossover — the first offspring (top one) takes its first five bits from the first parent and its remaining six bits from the second parent (as per the crossover mask).

The second offspring uses the same crossover mask, but switches the roles of the two parents.

crossover mask begins with a string containing n contiguous 1s, followed by the necessary no. of 0s to complete the string.

Two-point Crossover —

Crossover mask begins with n_0 zeros, followed by a contiguous string of n_1 ones, followed by the necessary no. of zeros to complete the string.

Uniform crossover — Crossover mask is generated as a random bit string with each bit chosen at random and independent of others.

Mutation — produces small random changes to the bit string by choosing a single bit at random, then changing its value.

2.3 Fitness Function and Selection

The fitness function defines the criterion
- for ranking potential hypotheses and
- for probabilistically selecting them for inclusion in the next-generation population.

In the algorithm GA, $Pr(h_i)$ is given as: (called Fitness proportionate selection or Roulette wheel selection)

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

(This is the ratio of the fitness of the hypothesis to the fitness of other members of the current population.)

3. An Illustrative Example

Although not guaranteed to find an optimal object, GAs often succeed in finding an object with high fitness.

Example system: GABIL

GABIL uses a GA to learn boolean concepts represented by a disjunctive set of propositional rules.

The algorithm used by GABIL is exactly the algorithm described earlier. It sets:

$r = 0.6$, $m = 0.001$ and

$p = 100$ to 1000 depending on the specific learning task.

The specific instantiation of the GA algorithm in GABIL can be summarized as follows:

(1) Representation

Each hypothesis in GABIL corresponds to a disjunctive set of propositional rules.

A set of rules is represented by concatenating the bit-string representations of individual rules.

Consider a hypothesis space in which rule preconditions are conjunctions of constraints over two boolean attributes a_1 and a_2.

The rule postcondition is described by a single bit (indicating True or False).

Thus, the hypothesis consisting of the two rules:

IF $a_1 = T \wedge a_2 = F$ THEN $c = T$;

IF $a_2 = T$ THEN $c = F$.

would be represented by the string

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

The length of the bit string grows with the no. of rules in the hypothesis.

This variable bit-string length requires a slight modification to the crossover operator, as described below.

(2) Genetic operators

GABIL uses the standard mutation operator (in which a single bit is chosen at random and replaced by its complement).

GABIL uses the crossover operator which is an extension to the two-point crossover operator. This approach

- accommodates the variable-length bit strings that encode rule sets, and
- constrains the system so that crossover occurs only between like sections of the bit strings.

To perform a crossover operation on two parents, two crossover points are first chosen at random in the first parent string.

Let d_1 (d_2) denote the distance from the leftmost (rightmost) of these two crossover points to the rule boundary immediately to its left.

The crossover points in the second parent are now randomly chosen, subject to the constraint that they must have the same d_1 and d_2 values.

Ex: If the two parent strings are

	a_1	a_2	c	a_1	a_2	c
h_1:	10	01	1	11	10	0

and

	a_1	a_2	c	a_1	a_2	c
h_2:	01	11	0	10	01	0

and the crossover points chosen for the first parent are the points following bit positions 1 and 8,

	a_1	a_2	c	a_1	a_2	c
h_1:	1[	01	1	11	]0	0

where [and] are crossover points, then $d_1 = 1$ and $d_2 = 3$.

Hence the allowed pairs of crossover points for the second parent include the pairs of bit positions <1,3>, <1,8> and <6,8>.

If the pair <1,3> is chosen,

	a_1	a_2	c		a_1	a_2	c
h_2 :	0	1	1	0	10	01	0

then the two resulting offspring will be

	a_1	a_2	c
h_3 :	11	10	0

and

	a_1	a_2	c		a_1	a_2	c		a_1	a_2	c
h_4 :	00	01	1		11	11	0		10	01	0

(3) Fitness function

The fitness of each hypothesized rule set is based on its classification accuracy over the training data.

The function is:

$$\text{Fitness}(h) = (\text{correct}(h))^2$$

where correct(h) is the percent of all training examples correctly classified by hypothesis h.

24.3.2015

4. Hypothesis Space Search

Population Evolution and the Schema Theorem:

Schema theorem characterizes the evolution over time of the population within a GA.

It is based on the concept of schemas or patterns that describe sets of bit strings.

schema — any string composed of 0s, 1s and *'s.

(* represents "don't care".)

Ex: schema 0*10 represents the set of bit strings that includes exactly 0010 and 0110.

- An individual bit string is a representative of each of the different schemas that it matches.

  Ex: 0010 is a representative of 2^4 distinct schemas including 00**, 0*10, ****, etc.

- A population of bit strings can be viewed in terms of the set of schemas that it represents and the no. of individuals associated with each of these schema.
- The schema theorem characterizes the evolution of the population in terms of the no. of instances representing each schema.

Suppose $m(s,t)$ — is the no. of instances of schema s in the population at time t (ie, during t^{th} generation)

The schema theorem describes the expected value of $m(s,t+1)$ in terms of $m(s,t)$, and other properties of the schema, population and GA algorithm parameters.

- The evolution of population depends on the :
selection step, recombination step and mutation step.

Consider the effect of selection step.

Suppose :

$f(h)$ — fitness of the individual bit string h.

$\bar{f}(t)$ — average fitness of all individuals in the population at time t.

n — total no. of individuals in the population

$h \in s \cap p_t$ — h is both a representative of schema s and a member of the population at time t.

$\hat{u}(s,t)$ — average fitness of instances of s at time t.

We calculate the expected value of $m(s,t+1)$ (denoted as : $E[m(s,t+1)]$).

- The probability distribution for selection is given as :

$$Pr(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{j=1}^{p} \text{Fitness}(h_j)}$$

Using this, we have

$$Pr(h) = \frac{f(h)}{\sum_{i=1}^{n} f(h_i)}$$

$$= \frac{f(h)}{n \cdot \bar{f}(t)}$$

- Now, if we select one member for the new population, then the probability that we will select a representative of schema s is

$$Pr(h \in s) = \sum_{h \in s \cap p_t} \frac{f(h)}{n \cdot \bar{f}(t)}$$

$$= \frac{\hat{u}(s,t)}{n \cdot \bar{f}(t)} \cdot m(s,t) \longrightarrow (1)$$

(the above step is from the fact that:

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cap p_t} f(h)}{m(s,t)}$$

ie, average-fitness $= \frac{\text{total-fitness}}{\text{no-of-instances}}$)

Therefore, the expected no. of instances of s resulting from the n independent selection steps that create the entire new generation is just n times the probability in (1).

$$\therefore E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} \cdot m(s,t)$$

5. Genetic Programming (GP)

GP is a form of evolutionary computation in which the individuals in the evolving population are "computer programs" rather than bit strings.

5.1 Representing programs

- Programs are represented by trees.
- Each function call is a node in the tree, and arguments to the function are its descendants.

Ex: Tree representation for the function $\sin(x) + \sqrt{x^2+y}$ is:

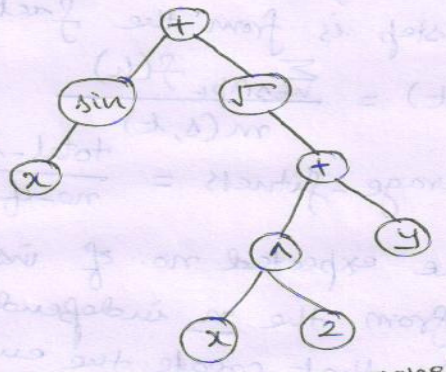
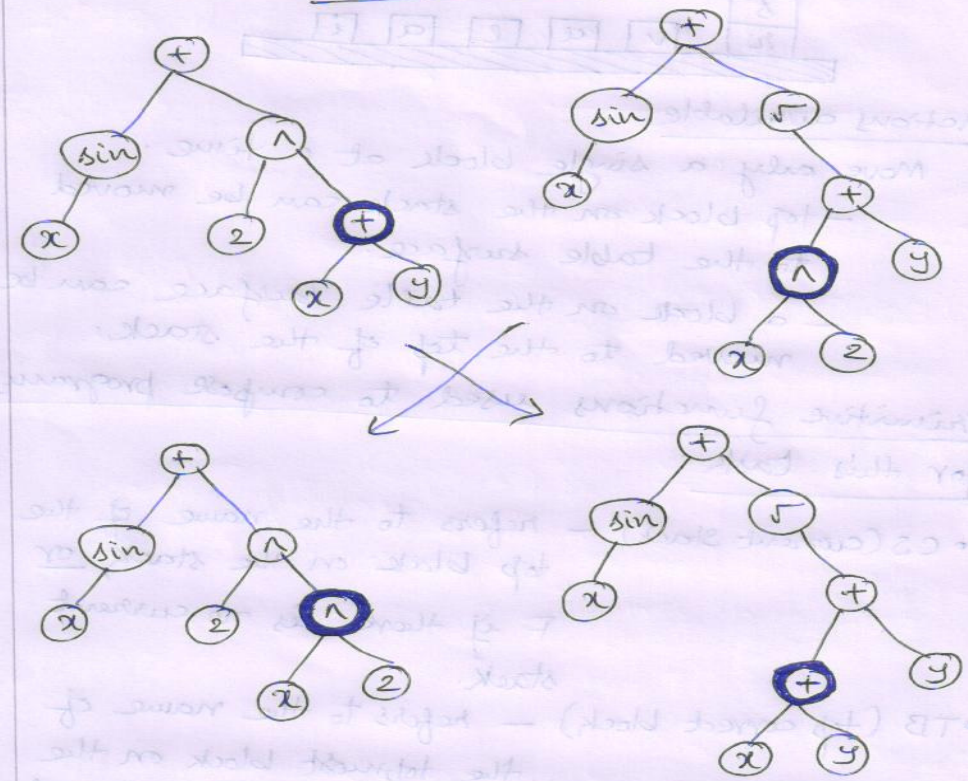


Fig. 1

Execution of a prototypical genetic programming algorithm:

- The algorithm maintains a population of individuals (in this case, program trees).
- On each iteration, it produces a new generation of individuals using selection, crossover and mutation.
- The fitness of a given individual program is determined by executing the program on a set of training data.
- Crossover operations are performed by replacing a randomly chosen subtree of one parent program by a subtree from the other parent program. (See Fig. 2)

Fig. 2 Crossover operation applied to two parent program trees

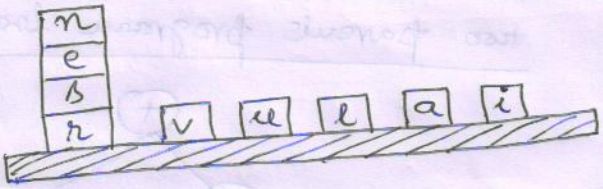


5.2 Illustrative Example

This is presented by Koza which involves learning an algorithm for stacking the blocks shown in Figure 3.

Task — develop a general algorithm for stacking the blocks into a single stack that spells the word "universal" independent of the initial configuration of blocks in the world.

Fig. 3 A block-stacking problem



Actions available

Move only a single block at a time.
- top block on the stack can be moved to the table surface.
- a block on the table surface can be moved to the top of the stack.

Primitive functions used to compose programs for this task:

- CS (current stack) — refers to the name of the top block on the stack, or F if there is no current stack.
- TB (top correct block) — refers to the name of the topmost block on the stack, such that it and those blocks beneath it are in the correct order.
- NN (Next Necessary) — refers to the name of the next block needed above TB in the stack, in order to spell the word "universal", or F if no more blocks are needed.

Other primitive functions:

- MS(x) (Move to Stack)
  if block x is on the table, then x is moved to the top of the stack and returns the value T.
  Otherwise, it does nothing and returns the value F.
- MT(x) (Move to Table)
  if x is on top of the stack, it is moved to the table and returns the value T.
  Otherwise, it returns the value F.
- (EQ x y) (EQual)
  returns T if x equals y, and returns F otherwise.
- (NOT x)
  returns T if x = F, and returns F if x = T.
- (DU x y) (Do Until)
  executes the expression repeatedly until expression y returns the value T.

To allow the system to evaluate the fitness of any given program, Koza provided a set of 166 training examples problems, representing a broad variety of initial block configurations, including problems of differing degrees of complexity.

The fitness of any given program was taken to be the no. of these examples solved by the algorithm.

The population was initialized to a set of 300 random programs.

After 10 generations, the system discovered the following program, which solves all 166 problems.

(EQ (DU (MT CS)(NOT CS))(DU (MS NN)(NOT NN)))

The first 'DU' repeatedly moves the current top of the stack onto the table, until the stack becomes empty.

The second 'DU' then repeatedly moves the next necessary block from the table onto the stack.

K. SRINIVASA RAO
Associate Professor
Dept. of CSE, GIT
GITAM University
Visakhapatnam

26-3-2015