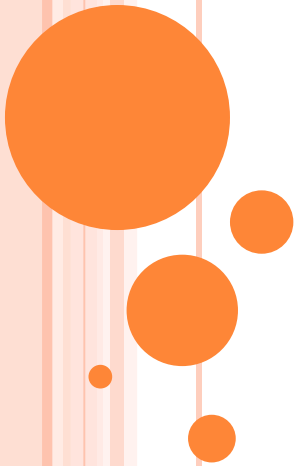
GENETIC ALGORITHMS



- Genetic algorithms provide an approach to learning that is based loosely on simulated evolution.

- Hypotheses are often described by bit strings whose interpretation depends on the application, though hypotheses may also be described by symbolic expressions or even computer programs.

- The search for an appropriate hypothesis begins with a population, or collection, of initial hypotheses.

- Members of the current population give rise to the next generation population by means of operations such as random mutation and crossover.

➢At each step, the hypotheses in the current population are evaluated relative to a given measure of fitness, with the most fit hypotheses selected probabilistically as seeds for producing the next generation.

➢They have been used to learn collections of rules for robot control and to optimize the topology and learning parameters for artificial neural networks.

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.

popularity of GAs is motivated by a number of factors including:

- Evolution is known to be a successful, robust method for adaptation within biological systems.
- GAs can search spaces of hypotheses containing complex interacting parts, where the impact of each part on overall hypothesis fitness may be difficult to model.
- Genetic algorithms are easily parallelized and can take advantage of the decreasing costs of powerful computer hardware.

Genetic algorithms and genetic programming are two of the more popular approaches in a field that is sometimes called evolutionary computation.

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.***

Example
If the task is to learn a strategy for playing chess, fitness could be defined as the number of games won by the individual when playing against other individuals in the current population.

GAs 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.

GA($Fitness$, $Fitness\_threshold$, p, r, m)

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

$Fitness\_threshold$: A threshold specifying the termination criterion.

p: The number 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 $[\max_h Fitness(h)] < Fitness\_threshold$ do

  Create a new generation, P_S:

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

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

  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.

A prototypical genetic algorithm is described in Table 9.1. The inputs to this algorithm include the fitness function for ranking candidate hypotheses, a threshold defining an acceptable level of fitness for terminating the algorithm, the size of the population to be maintained, and parameters that determine how successor populations are to be generated: the fraction of the population to be replaced at each generation and the mutation rate.

First, a certain number of hypotheses from the current population are selected for inclusion in the next generation. These are selected ***probabilistically, where the probability of selecting*** hypothesis ***hi is given by***

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

Thus, the probability that a hypothesis will be selected is proportional to its own fitness and is inversely proportional to the fitness of the other competing hypotheses in the current population.

Additional members are generated using a crossover operation.

takes two parent hypotheses from the current generation and creates two offspring hypotheses by recombining portions of both parents.

Representing Hypotheses

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

For example, sets of if-then rules can easily be represented in this way, by choosing an encoding of rules that allocates specific substrings for each rule precondition and postcondition.

Example

Outlook→ Sunny, Overcast, Rainy

One obvious way to represent a constraint on ***Outlook is to use a bit string of length three, in which each bit position corresponds*** to one of its three possible values.

Placing a 1 in some position indicates that the attribute is allowed to take on the corresponding value.

010 → *Outlook must take on the second of these values,* or ***Outlook = Overcast.***
011 → $(Outlook = Overcast \vee Rain)$

111→ we don't care which of its possible values the attribute takes on.

For example, consider a second attribute, ***Wind, that can take on the value Strong or Weak. A rule precondition such as***

$$(Outlook = Overcast \vee Rain) \wedge (Wind = Strong)$$

can then be represented by the following bit string of length five:

Outlook	*Wind*
011	10

Rule postconditions (such as ***PlayTennis = yes)***

IF $Wind = Strong$ THEN $PlayTennis = yes$

would be represented by the string

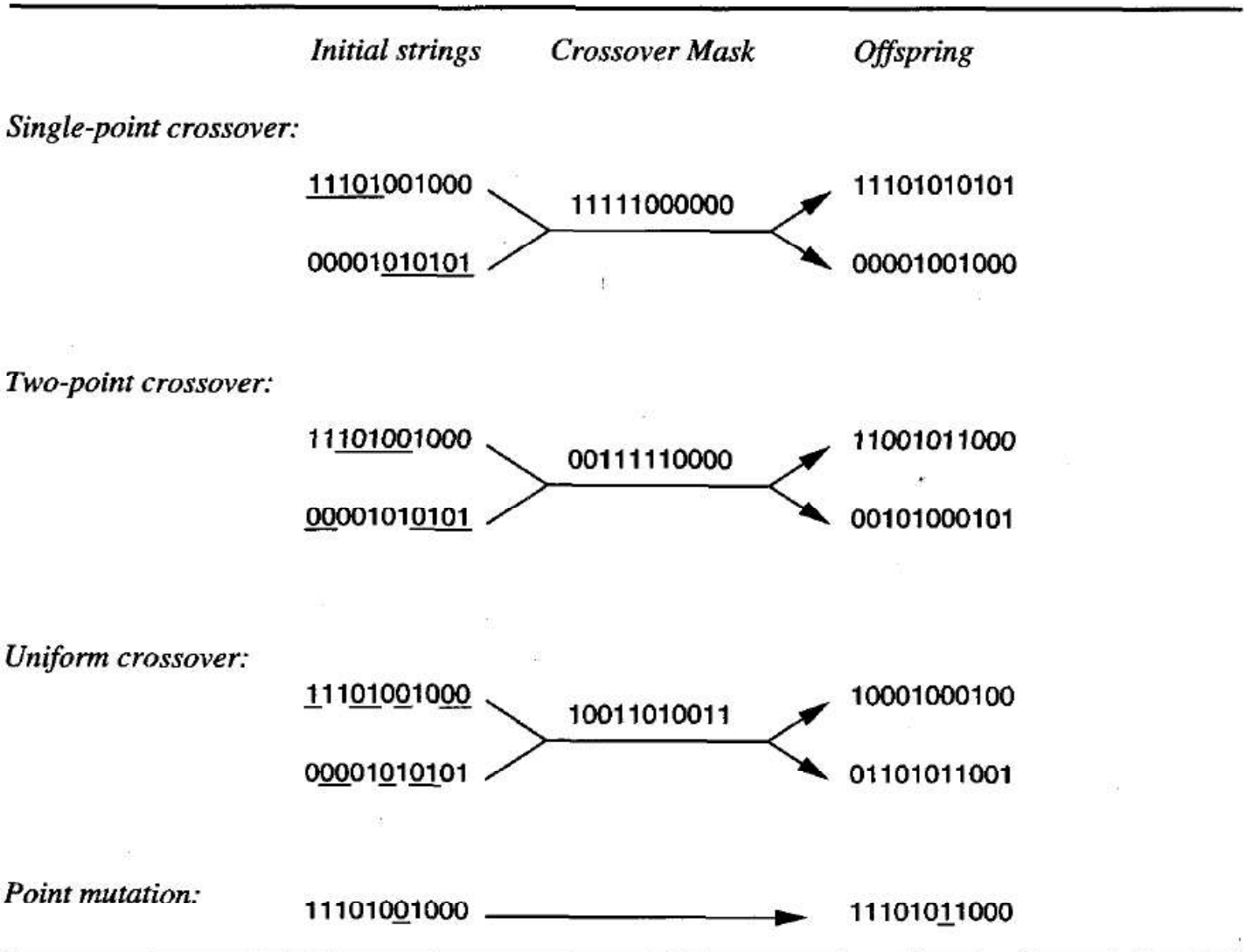
Outlook	*Wind*	*PlayTennis*
111	10	10

note in the rule encoding in the above paragraph the bit string 11 1 10 11 represents a rule whose postcondition does not constrain the target attribute ***PlayTennis.***

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.

Typical GA operators for manipulating bit string hypotheses


Initial strings
Crossover Mask
Offspring
Single-point crossover:
11101001000
00001010101
11111000000
11101010101
00001001000
Two-point crossover:
11101001000
00001010101
00111110000
11001011000
00101000101
Uniform crossover:
11101001000
00001010101
10011010011
10001000100
01101011001
Point mutation:
11101001000
11101011000

Fitness Function and Selection

The probability that a hypothesis will be selected is given by the ratio of its fitness to the fitness of other members of the current population is sometimes called ***Fitness proportionate selection, or roulette wheel selection.***

tournament selection, two hypotheses are first chosen at random from the current population. With some predefined probability p the more fit of these two is then selected, and with probability (1 - p) the less fit hypothesis is selected.

In another method called ***rank selection, the*** hypotheses in the current population are first sorted by fitness. The probability that a hypothesis will be selected is then proportional to its rank in this sorted list, rather than its fitness.

AN ILLUSTRATIVE EXAMPLE

To illustrate the use of GAS for concept learning, we briefly summarize the GABIL system described by DeJong et al. (1993).

Representation

value of the target attribute c. 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

Genetic operators.

	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[0	01	1	11	1]0	0

where "[" and "]" indicate 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 $\langle 1, 3\rangle$, $\langle 1, 8\rangle$, and $\langle 6, 8\rangle$. If the pair $\langle 1, 3\rangle$ happens to be chosen,

	a_1	a_2	c	a_1	a_2	c
h_2 :	0[1	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

- **Fitness function.** The fitness of each hypothesized rule set is based on its classification accuracy over the training data. In particular, the function used to measure fitness is

$$Fitness(h) = (correct(h))^2$$

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

HYPOTHESIS SPACE SEARCH

Population Evolution and the Schema Theorem

It is based on the concept of ***schemas, or patterns***

terns that describe sets of bit strings. To be precise, a schema is any string composed of 0s, 1s, and *'s. Each schema represents the set of bit strings containing the indicated 0s and 1s, with each "*" interpreted as a "don't care." For example, the schema 0*10 represents the set of bit strings that includes exactly 0010 and 0110.

The schema theorem characterizes the evolution of the population within a GA in terms of the number of instances representing each schema. Let $m(s, t)$ denote the number of instances of schema s in the population at time t (i.e., during the tth 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.

Let $f(h)$ denote the fitness of the individual bit string h and $\bar{f}(t)$ denote the average fitness of all individuals in the population at time t. Let n be the total number of individuals in the population. Let $h \in s \cap p_t$ indicate that the individual h is both a representative of schema s and a member of the population at time t. Finally, let $\hat{u}(s,t)$ denote the average fitness of instances of schema s in the population at time t.

We are interested in calculating the expected value of $m(s,t+1)$, which we denote $E[m(s,t+1)]$. We can calculate $E[m(s,t+1)]$ using the probability distribution for selection given in Equation (9.1), which can be restated using our current terminology as follows:

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

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

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

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

$$= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t) \tag{9.2}$$

The second step above follows from the fact that by definition,

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

Equation (9.2) gives the probability that a single hypothesis selected by the GA will be an instance of schema s. Therefore, the expected number of instances of s resulting from the n independent selection steps that create the entire new generation is just n times this probability.

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \tag{9.3}$$

GENETIC PROGRAMMING

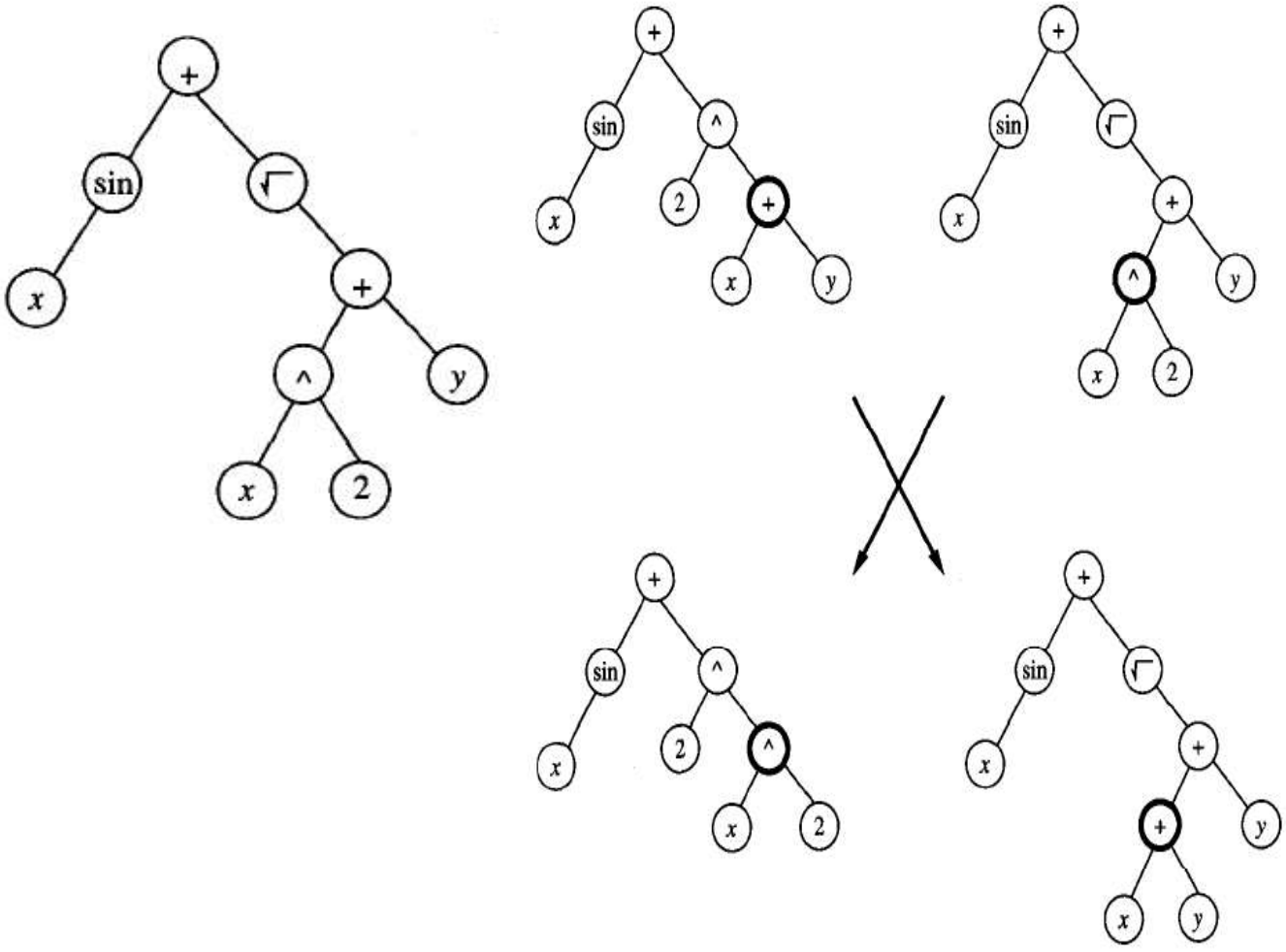
Genetic programming (GP) is a form of evolutionary computation in which the individuals in the evolving population are computer programs rather than bit strings.

Representing Programs

Programs manipulated by a GP are typically represented by trees corresponding to the parse tree of the program.

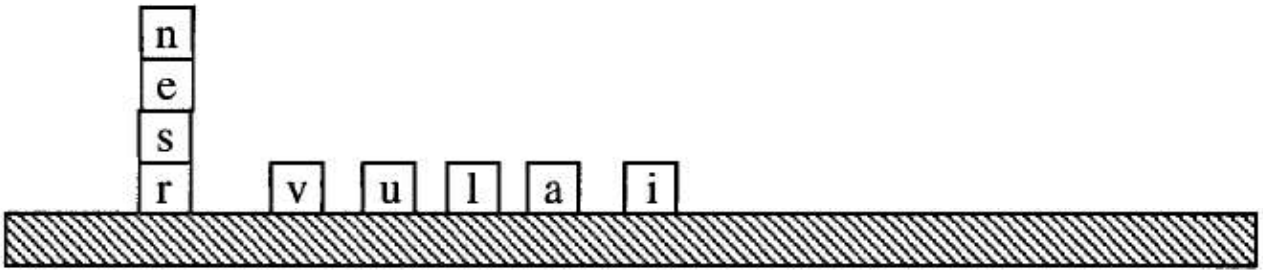
Each function call is represented by a node in the tree, and the arguments to the function are given by its descendant nodes.

$$\sin(x) + \sqrt{x^2 + y}.$$


+
sin
x
+
^
y
x
2
+
sin
^
x
2
+
x
y
+
sin
x
+
^
y
x
2
+
sin
^
x
2
+
x
y
+
sin
x
+
+
y
x
y

Illustrative Example

One illustrative example presented by Koza (1992) involves learning an algorithm for stacking the blocks shown in Figure 9.3.



primitive functions used to compose programs for this task include the following three terminal arguments:

- CS (current stack), which refers to the name of the top block on the stack, or *F* if there is no current stack.
- TB (top correct block), which 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), which 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.

In addition to these terminal arguments, the program language in this application included the following primitive functions:

- (MS x) (move to stack), if block x is on the table, this operator moves x 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 block x is somewhere in the stack, this moves the block at the top of the stack to the table and returns the value T. Otherwise, it returns the value F.
- (EQ x y) (equal), which returns T if x equals y, and returns F otherwise.
- (NOT x), which returns T if $x = F$, and returns F if $x = T$.
- (DU x y) (do until), which executes the expression x repeatedly until expression y returns the value T.

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

PARALLELIZING GENETIC ALGORITHMS

GAs are naturally suited to parallel implementation, and a number of approaches to parallelization have been explored. *Coarse grain* approaches to parallelization subdivide the population into somewhat distinct groups of individuals, called *demes*. Each deme is assigned to a different computational node, and a standard GA search is performed at each node. Communication and cross-fertilization between demes occurs on a less frequent basis than within demes. Transfer between demes occurs by a *migration* process, in which individuals from one deme are copied or transferred to other demes. This process is modeled after the kind of cross-fertilization that might occur between physically separated subpopulations of biological species. One benefit of such approaches is that it reduces the crowding problem often encountered in nonparallel GAs, in which the system falls into a local optimum due to the early appearance of a genotype that comes to dominate the entire population. Examples of coarse-grained parallel GAs are described by Tanese (1989) and by Cohoon et al. (1987).