Genetic Algorithms

[Read Chapter 9]
[Exercises 9.1, 9.2, 9.3, 9.4]

- Evolutionary computation
- Prototypical GA
- An example: GABIL
- Genetic Programming
- Individual learning and population evolution

Evoluationary Computation

1. Computational procedures patterned after biological evolution
2. Search procedure that probabilistically applies search operators to set of points in the search space

Biological Evolution

Lamarck and others:

- Species "transmute" over time

Darwin and Wallace:

- Consistent, heritable variation among individuals in population
- Natural selection of the fittest

Mendel and genetics:

- A mechanism for inheriting traits
- genotype $\rightarrow$ phenotype mapping

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

- *Initialize:* $P \leftarrow p$ random hypotheses
- *Evaluate:* for each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$
  1. *Select:* Probabilistically select $(1 - r)p$ members of P to add to P_S.
  $$\Pr(h_i) = \frac{Fitness(h_i)}{\Sigma_{j=1}^{p} Fitness(h_j)}$$
  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P. 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:* Invert a randomly selected bit in $m \cdot p$ random members of P_s
  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.

Representing Hypotheses

Represent

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

by

$$\begin{array}{cc} Outlook & Wind \\ 011 & 10 \end{array}$$

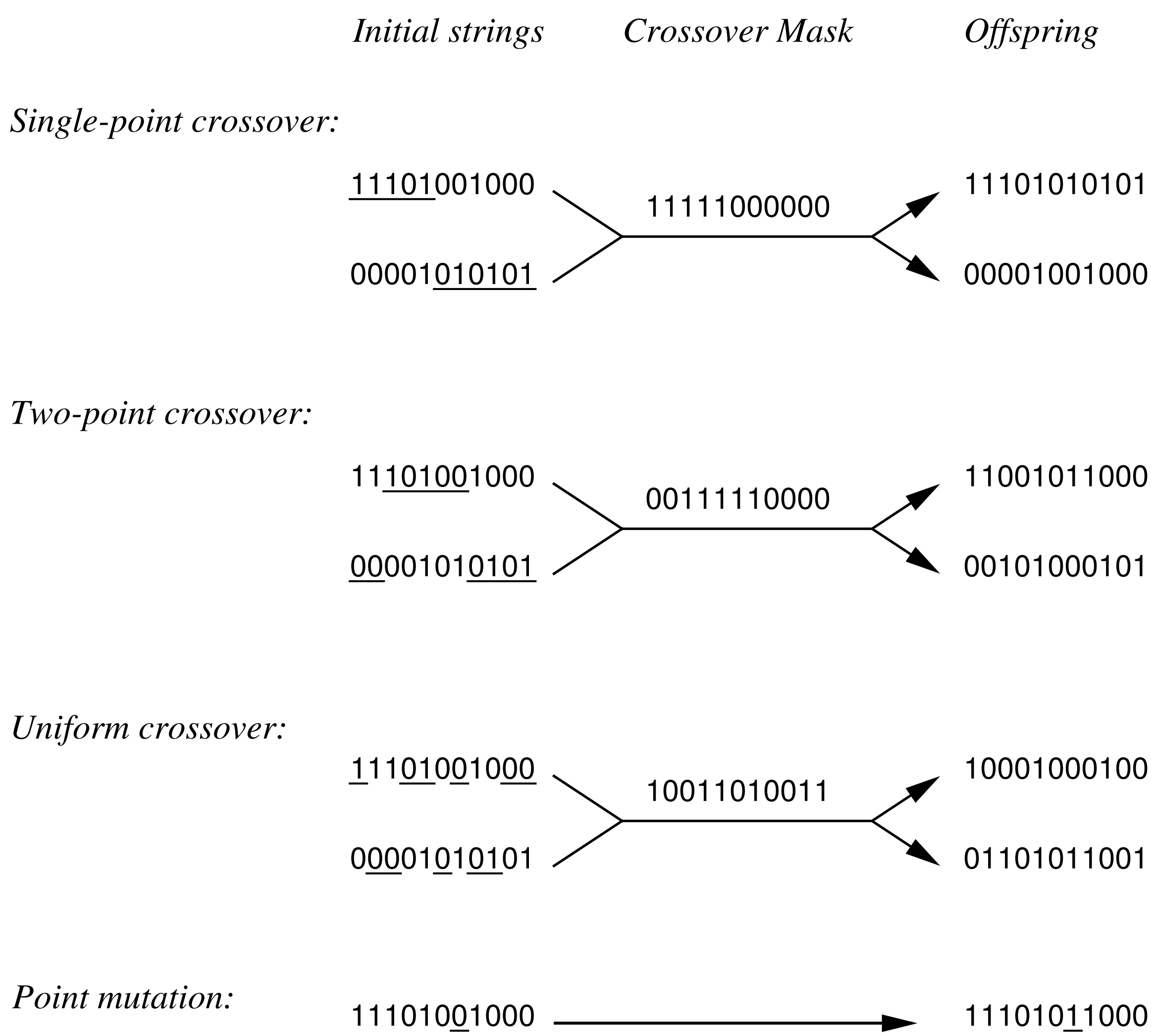
Represent

$$\text{IF} \quad Wind = Strong \quad \text{THEN} \quad PlayTennis = yes$$

by

$$\begin{array}{ccc} Outlook & Wind & PlayTennis \\ 111 & 10 & 10 \end{array}$$

Operators for Genetic Algorithms



Selecting Most Fit Hypotheses

Fitness proportionate selection:

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

... can lead to *crowding*

Tournament selection:

- Pick h_1, h_2 at random with uniform prob.
- With probability p, select the more fit.

Rank selection:

- Sort all hypotheses by fitness
- Prob of selection is proportional to rank

GABIL [DeJong et al. 1993]

Learn disjunctive set of propositional rules, competitive with C4.5

Fitness:

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

Representation:

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

represented by

$$\begin{array}{cccccc} a_1 & a_2 & c & a_1 & a_2 & c \\ 10 & 01 & 1 & 11 & 10 & 0 \end{array}$$

Genetic operators: ???

- want variable length rule sets
- want only well-formed bitstring hypotheses

Crossover with Variable-Length Bitstrings

Start with

$$
\begin{array}{rccccccc}
 & a_1 & a_2 & c & & a_1 & a_2 & c \\
h_1 : & 10 & 01 & 1 & & 11 & 10 & 0 \\
 & & & & & & & \\
h_2 : & 01 & 11 & 0 & & 10 & 01 & 0
\end{array}
$$

1. choose crossover points for h_1, e.g., after bits 1, 8
2. now restrict points in h_2 to those that produce bitstrings with well-defined semantics, e.g., $\langle 1, 3 \rangle$, $\langle 1, 8 \rangle$, $\langle 6, 8 \rangle$.

if we choose $\langle 1, 3 \rangle$, result is

$$
\begin{array}{rccc}
 & a_1 & a_2 & c \\
h_3 : & 11 & 10 & 0
\end{array}
$$

$$
\begin{array}{rccccccccccc}
 & a_1 & a_2 & c & & a_1 & a_2 & c & & a_1 & a_2 & c \\
h_4 : & 00 & 01 & 1 & & 11 & 11 & 0 & & 10 & 01 & 0
\end{array}
$$

GABIL Extensions

Add new genetic operators, also applied probabilistically:

1. *AddAlternative*: generalize constraint on a_i by changing a 0 to 1
2. *DropCondition*: generalize constraint on a_i by changing every 0 to 1

And, add new field to bitstring to determine whether to allow these

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

So now the learning strategy also evolves!

GABIL Results

Performance of GABIL comparable to symbolic rule/tree learning methods C4.5, ID5R, AQ14

Average performance on a set of 12 synthetic problems:

- GABIL without AA and DC operators: 92.1% accuracy
- GABIL with AA and DC operators: 95.2% accuracy
- symbolic learning methods ranged from 91.2 to 96.6

Schemas

How to characterize evolution of population in GA?

Schema = string containing 0, 1, * ("don't care")

- Typical schema: 10**0*
- Instances of above schema: 101101, 100000, ...

Characterize population by number of instances representing each possible schema

- $m(s, t)$ = number of instances of schema s in pop at time t

Consider Just Selection

- $\bar{f}(t)$ = average fitness of pop. at time t
- $m(s,t)$ = instances of schema s in pop at time t
- $\hat{u}(s,t)$ = ave. fitness of instances of s at time t

Probability of selecting h in one selection step

$$\begin{aligned} \Pr(h) &= \frac{f(h)}{\sum_{i=1}^{n} f(h_i)} \\ &= \frac{f(h)}{n\bar{f}(t)} \end{aligned}$$

Probabilty of selecting an instance of s in one step

$$\begin{aligned} \Pr(h \in s) &= \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)} \\ &= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t) \end{aligned}$$

Expected number of instances of s after n selections

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

Schema Theorem

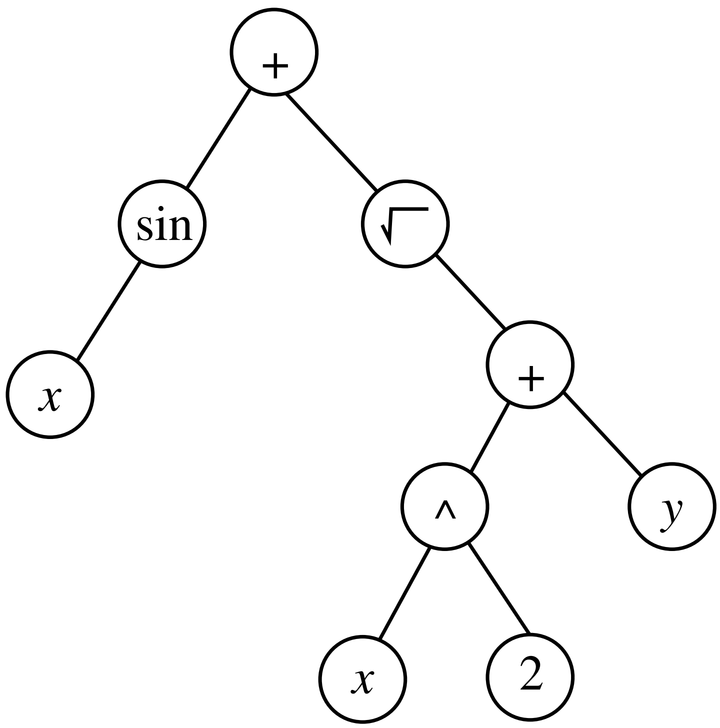
$$E[m(s,t{+}1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right) (1{-}p_m)^{o(s)}$$

- $m(s,t)$ = instances of schema s in pop at time t
- $\bar{f}(t)$ = average fitness of pop. at time t
- $\hat{u}(s,t)$ = ave. fitness of instances of s at time t
- p_c = probability of single point crossover operator
- p_m = probability of mutation operator
- l = length of single bit strings
- $o(s)$ number of defined (non "*") bits in s
- $d(s)$ = distance between leftmost, rightmost defined bits in s

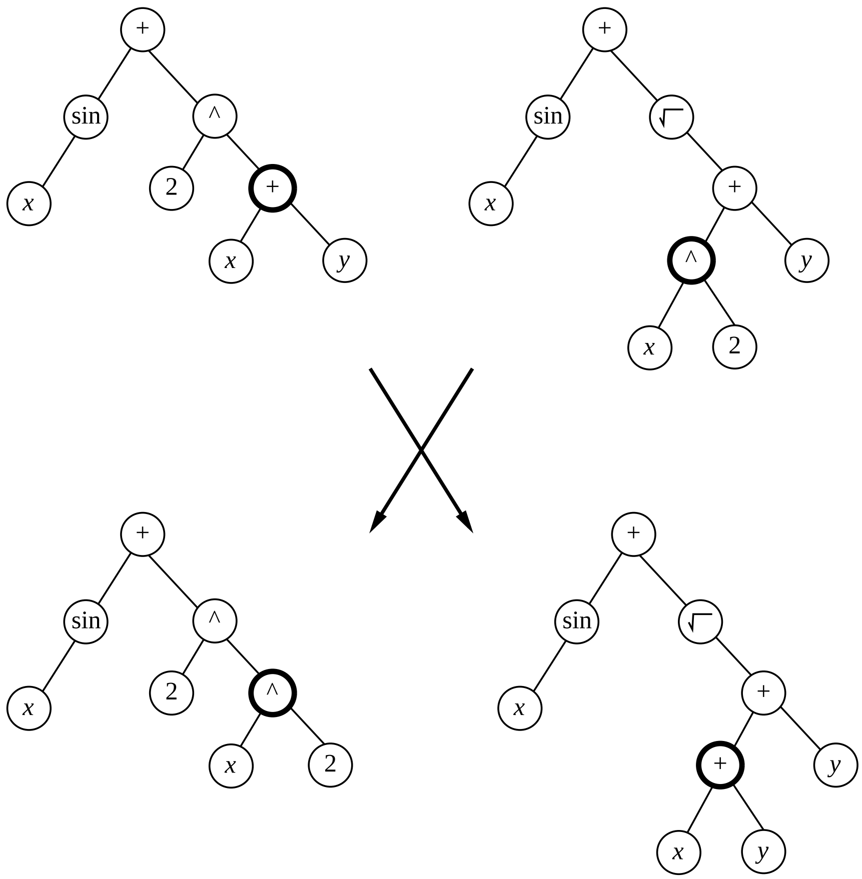
Genetic Programming

Population of programs represented by trees

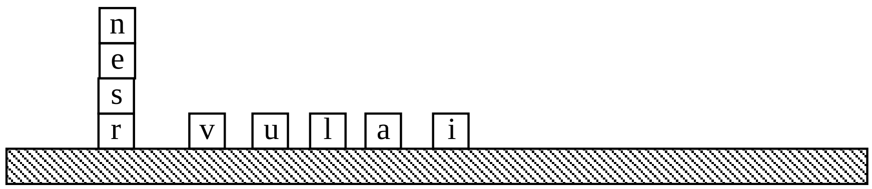
$$\sin(x) + \sqrt{x^2 + y}$$



Crossover



Block Problem



Goal: spell UNIVERSAL

Terminals:

- CS ("current stack") = name of the top block on stack, or F.
- TB ("top correct block") = name of topmost correct block on stack
- NN ("next necessary") = name of the next block needed above TB in the stack

Primitive functions:

- (MS x): ("move to stack"), if block x is on the table, moves x to the top of the stack and returns the value T. Otherwise, does nothing and returns the value F.
- (MT x): ("move to table"), if block x is somewhere in the stack, moves the block at the top of the stack to the table and returns the value T. Otherwise, returns F.
- (EQ x y): ("equal"), returns T if x equals y, and returns F otherwise.
- (NOT x): returns T if $x = F$, else returns F
- (DU x y): ("do until") executes the expression x repeatedly until expression y returns the value T

Learned Program

Trained to fit 166 test problems

Using population of 300 programs, found this after 10 generations:

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

Genetic Programming

More interesting example: design electronic filter circuits

- Individuals are programs that transform begining circuit to final circuit, by adding/subtracting components and connections
- Use population of 640,000, run on 64 node parallel processor
- Discovers circuits competitive with best human designs