Genetic Algorithms

Inspired by biological processes that produce genetic change in populations of individuals.

Genetic algorithms (GAs) are adaptive search procedures that usually include three basic elements:

1. A Darwinian notion of fitness: the most fit individuals have the best chance of survival and reproduction.
2. Mating operators: individuals contribute their genetic material to their children.
3. Mutation: individuals are subject to random changes in their genetic material.

Slide CS478–1

Learning through populations

- Many learning algorithms commit to a single hypothesis at any one point in time.
- Genetic algorithms maintain a population of hypotheses.
- Each hypothesis is evaluated using a **fitness function**. The fitness scores force individuals to compete for the privilege of survival and reproduction.
- Genetic algorithms are typically performance-oriented. The fitness of a hypothesis is often measured by the performance of the hypothesis on a set of tasks.

Slide CS478–2

Genetic algorithms as search

- Genetic algorithms are local heurisitc search algorithms.
- "Weak" (i.e. general-purpose) method.
- Especially good for problems that have large and poorly understood search spaces.
- Genetic algorithms use a randomized parallel beam search to explore the hypothesis space.
- You must be able to define a good fitness function, and of course, a good hypothesis representation.

Slide CS478–3

Binary string representations

- Hypotheses are usually represented using bit strings.
- Hypotheses represented can be arbitrarily complex.
- E.g. each attribute is allocated a specific portion of the string, which encodes the attribute values that are acceptable.
- Each bit encodes whether a single attribute value is acceptable or not. So you need N bits to represent N attribute values.
- Why not use binary-valued encoding (e.g., 2 bits could represent 4 values)?
- Bit string representation allows crossover operation to change multiple values. Crossover and mutation can also

Slide CS478–4

produce previously unseen values.

Slide CS478–5

Representing Hypotheses

Bit sequences can also represent conjunctions of constraints on attribute values. For example:

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

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

Bit sequences can also represent rules, or more complicated structures. For example:

$$\text{IF} \;\; Wind = Strong \;\;\; \text{THEN} \;\; Ski? = yes$$

Slide CS478–6

	$Outlook$	$Wind$	$Ski?$
$\Rightarrow$	011	10	1

Slide CS478–7

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

- $P \leftarrow$ randomly generate p hypotheses
- For each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$
  1. Probabilistically **select** $(1-r)p$ members of P to add to P_S.
  2. Probabilistically choose $\frac{r \cdot p}{2}$ pairs of hypotheses from P. For each pair, $\langle h_1, h_2 \rangle$, apply **crossover** and add the offspring to P_s
  3. **Mutate** $m \cdot p$ random members of P_s
  4. $P \leftarrow P_s$
  5. For each h in P, compute $Fitness(h)$
- Return the hypothesis in P with the highest fitness.

Slide CS478–8

Selecting Most Fit Hypotheses

Hypotheses are chosen probabilistically for survival and crossover based on **fitness proportionate selection**:

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

Other selection methods include:

- **Tournament Selection:** 2 hypotheses selected at random. With probability p, the most fit is selected. With probability $(1 - p)$, the less fit is selected.

- **Rank Selection:** The hypotheses are sorted by fitness and the probability of selecting a hypothesis is proportional to its rank in the list.

Crossover Operators

Single-point crossover:

Parent A:	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
Parent B:	0	1	0	1	1	1	0	1	1	0
Child AB:	**1**	**0**	**0**	**1**	**0**	1	0	1	1	0
Child BA:	0	1	0	1	1	**1**	**1**	**1**	**0**	**1**

Two-point crossover:

Parent A:	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
Parent B:	0	1	0	1	1	1	0	1	1	0
Child AB:	**1**	**0**	**0**	**1**	1	1	0	**1**	**0**	**1**
Child BA:	0	1	0	1	**0**	**1**	**1**	1	1	0

Uniform Crossover

Uniform crossover:

Parent A:	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
Parent B:	0	1	0	1	1	1	0	1	1	0
Child AB:	**1**	1	**0**	**1**	1	1	**1**	1	**0**	**1**
Child BA:	0	**0**	0	1	**0**	**1**	0	**1**	1	0

Mutation

Mutation: randomly toggle one bit

Individual A:	1	0	0	1	0	1	1	1	0	1
Individual A':	1	0	0	**0**	0	1	1	1	0	1

Mutation

- The **mutation** operator introduces random variations, allowing hypotheses to jump to different parts of the search space.
- What happens if the mutation rate is too low?
- What happens if the mutation rate is too high?
- A common strategy is to use a high mutation rate when learning begins but to decrease the mutation rate as learning progresses.

Learning illegal structures

Consider the traveling salesman problem, where an individual represents a potential solution. The standard crossover operator can produce illegal children:

Parent A:	ITH	Pitt	Chicago	Denver	Boise
Parent B:	Boise	Chicago	ITH	Phila	Pitt
Child AB:	ITH	Pitt	Chicago	Phila	Pitt
Child BA:	Boise	Chicago	ITH	Denver	Boise

Two solutions:

1. define special genetic operators that only produce syntactically and semantically legal hypotheses.
2. ensure that the fitness function returns extremely low fitness values to illegal hypotheses.

Slide CS478–17

Applications: Parameter Optimization

- Parameter optimization problems are well-suited for GAs. Each individual represents a set of parameter values and the GA tries to find the set of parameter values that achieves the best performance.
- The crossover operator creates new combinations of parameter values and, using a binary representation, both the crossover and mutation operators can produce new values.
- Many learning systems can be recast as parameter optimization problems. For example, most neural networks use a fixed architecture so learning consists entirely of adjusting weights and thresholds.

Slide CS478–18

GABIL [DeJong et al. 1993]

Learn disjunctive set of propositional rules **Fitness:**

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

Representation:

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

represented by

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

Genetic operators: ???

Slide CS478–19

Crossover with Variable-Length Bitstrings

Start with

	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

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., $\langle 1, 3\rangle$, $\langle 1, 8\rangle$, $\langle 6, 8\rangle$.

Slide CS478–20

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

	a_1	a_2	c
h_3 :	11	10	0

	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

Slide CS478–21

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

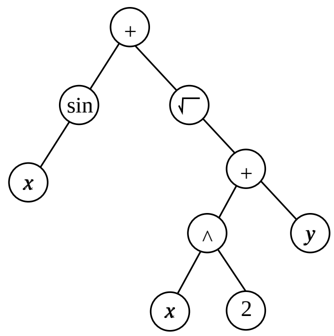
So now the learning strategy also evolves!

Slide CS478–22

Genetic Programming

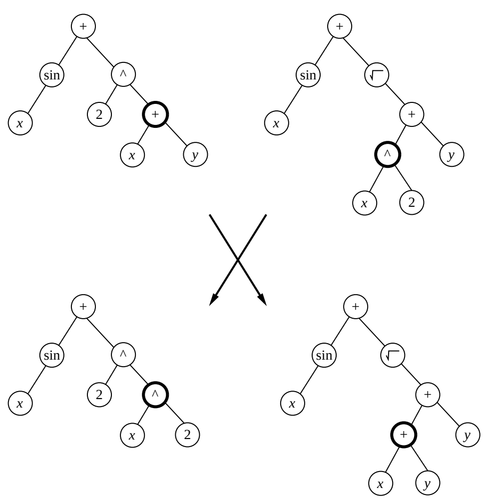
In **Genetic Programming**, programs are evolved instead of bit strings. Programs are often represented by trees. For example:

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



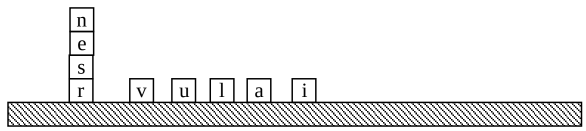
Slide CS478–23

Crossover in genetic programming



Slide CS478–24

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

Slide CS478–25

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

Slide CS478–26

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

Biological Evolution

Lamarck (19th century)

- Believed individual genetic makeup was altered by lifetime experience
- But current evidence contradicts this view

What is the impact of individual learning on population evolution?

Baldwin Effect

Assume

- Individual learning has no direct influence on individual DNA
- But ability to learn reduces need to "hard wire" traits in DNA – can perform local search!

Then

- Ability of individuals to learn will support more diverse gene pool
- More diverse gene pool will support faster evolution of gene pool

→ individual learning (indirectly) increases rate of evolution

Computer Experiments on Baldwin Effect

[Hinton and Nowlan, 1987]

Evolve simple neural networks:

- Some network weights fixed during lifetime, others trainable
- Genetic makeup determines which are fixed, and their weight values

Results:

- With no individual learning, population failed to improve over time
- When individual learning allowed

Slide CS478–31

  - Early generations: population contained many individuals with many trainable weights
  - Later generations: higher fitness, while number of trainable weights decreased

Slide CS478–32