Lecture
Genetic Programming

CIS 412 Artificial Intelligence
Umass, Dartmouth

Genetic programming

- One of the central problems in computer science is how to make computers solve problems without being explicitly programmed to do so.
- Genetic programming offers a solution through the evolution of computer programs by methods of natural selection.
- In fact, genetic programming is an extension of the conventional genetic algorithm, but the goal of genetic programming is not just to evolve a bitstring representation of some problem but the computer code that solves the problem.

GP

- Genetic programming is a recent development in the area of evolutionary computation. It was greatly stimulated in the 1990s by **John Koza**.
- According to Koza, genetic programming searches the space of possible computer programs for a program that is highly fit for solving the problem at hand.
- Any computer program is a sequence of operations (functions) applied to values (arguments), but different programming languages may include different types of statements and operations, and have different syntactic restrictions.

LISP as a GP language

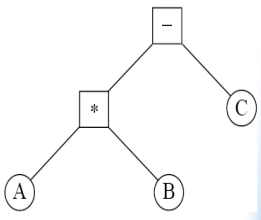
- GP manipulates programs by applying genetic operators.
- Programming language should permit a computer program to be manipulated as data and the newly created data to be executed as a program.
- **LISP** was chosen as the main language for genetic programming.

LISP structure

- LISP has a highly symbol-oriented structure. Its basic data structures are **atoms** and **lists**.
- An atom is the smallest indivisible element of the LISP syntax. The number *21*, the symbol *X* and the string *"This is a string"* are examples of LISP atoms.
- A list is an object composed of atoms and/or other lists. LISP lists are written as an ordered collection of items inside a pair of parentheses.

S-expressions

- (− (* A B) C)
- Both atoms and lists are called symbolic expressions or **S-expressions**. In LISP, all data and all programs are S-expressions.
- This gives LISP the ability to operate on programs as if they were data.
- Any LISP S-expression can be depicted as a rooted point-labeled tree with ordered branches.



GP algorithm steps

- *Five preparatory steps*:
- 1. Determine the set of terminals.
- 2. Select the set of primitive functions.
- 3. Define the fitness function.
- 4. Decide on the parameters for controlling the run.
- 5. Choose the method for designating a result of
- the run.

GP example

Pythagores' Theorem

The theorem says that the hypotenuse, c, of a right triangle with short sides a and b is given by

$$c = \sqrt{a^2 + b^2}.$$

Aim of GP

The aim of genetic programming is to find a program that matches with this theorem.

Performance measure - fitness

- We use a number of different **fitness cases**. The fitness cases for the Pythagorean Theorem are represented by the samples of right triangles in Table. These fitness cases are chosen at random over a range of values of variables *a* and *b*.

Side *a*	Side *b*	Hypotenuse *c*	Side *a*	Side *b*	Hypotenuse *c*
3	5	5.830952	12	10	15.620499
8	14	16.124515	21	6	21.840330
18	2	18.110770	7	4	8.062258
32	11	33.837849	16	24	28.844410
4	3	5.000000	2	9	9.219545

GP algorithm steps

Step 1: Determine the set of terminals. The terminals correspond to the inputs of the computer program to be discovered. Our program takes two inputs, a and b.

Step 2: Select the set of primitive functions. The functions can be presented by standard arithmetic operations, standard programming operations, standard mathematical functions, logical functions or domain-specific functions. Our program will use four standard arithmetic operations $+$, $-$, $*$ and $\div$, and one mathematical function $\sqrt{}$.

GP algorithm steps

Define the fitness function. A fitness function evaluates how well a particular computer program can solve the problem. For our problem, the fitness of the computer program can be measured by the error between the actual result produced by the program and the correct result given by the fitness case.

GP algorithm steps

Step 4: Decide on the parameters for controlling the run. For controlling a run, genetic programming uses the same primary parameters as those used for GAs. They include the population size and the maximum number of generations to be run.

Step 5: Choose the method for designating a result of the run. It is common practice in genetic programming to designate the best-so-far generated program as the result of a run.

GP run

Run

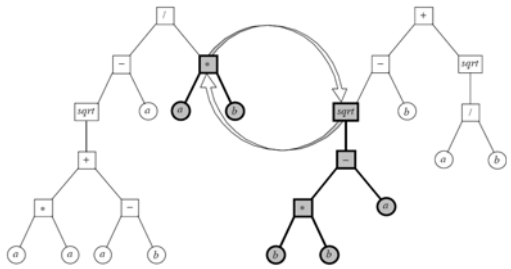
Once these five steps are complete, a run can be made.

- The run of genetic programming starts with a random generation of an initial population of computer programs.
- Each program is composed of functions $+, -, *, \div$ and $\sqrt{}$, and terminals a and b.

In the initial population, all computer programs usually have poor fitness, but some individuals are more fit than others.

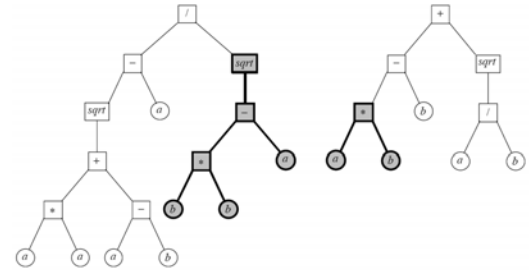
Just as a fitter chromosome is more likely to be selected for reproduction, so a fitter computer program is more likely to survive by copying itself into the next generation.

Two parental s-expressions - crossover



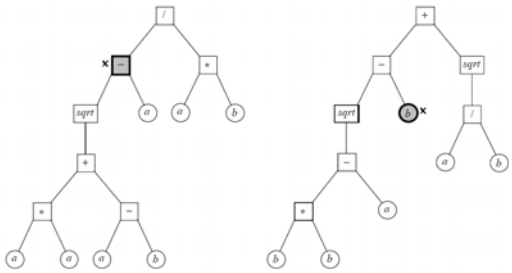
(/ (− (sqrt (+ (∗ a a) (− a b))) a) (∗ a b)) (+ (− (sqrt (− (∗ b b) a)) b) (sqrt (/ a b)))

Two offspring s-expressions



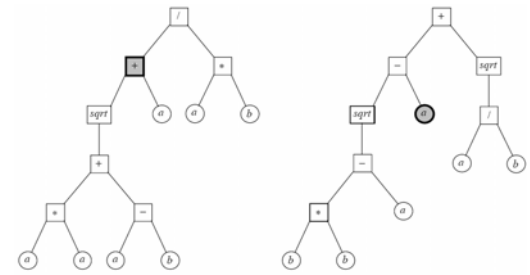
(/ (− (sqrt (+ (∗ a a) (− a b))) a) (sqrt (− (∗ b b) a))) (+ (−(∗ a b) b) (sqrt (/ a b)))

Mutation in GP - original



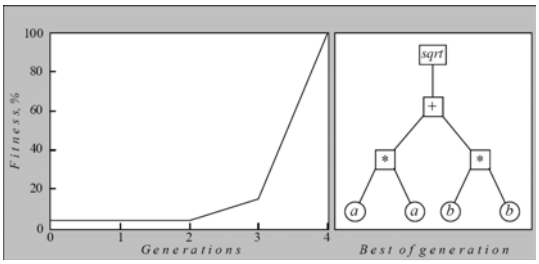
(/ (− (sqrt (+ (∗ a a) (− a b))) a) (∗ a b)) (+ (− (sqrt (− (∗ b b) a)) b) (sqrt (/ a b)))

Mutation in GP - result



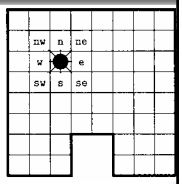
(/ (+ (sqrt (+ (∗ a a) (− a b))) a) (∗ a b)) (+ (− (sqrt (− (∗ b b) a)) a) (sqrt (/ a b)))

Fitness history of the best s-expression

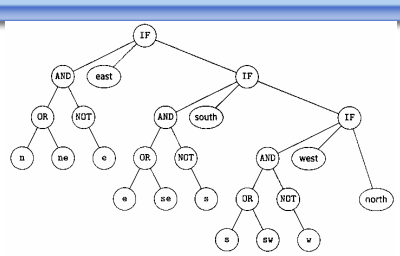


Example: Wall-Following Robot

- Program Representation in GP
  - Functions
    - AND (x, y) = 0 if x = 0; else y
    - OR (x, y) = 1 if x = 1; else y
    - NOT (x) = 0 if x = 1; else 1
    - IF (x, y, z) = y if x = 1; else z
  - Terminals
    - Actions: move the robot one cell to each direction {north, east, south, west}
    - Sensory input: its value is 0 whenever the corresponding cell is free for the robot to occupy; otherwise, 1.
      - {n, ne, e, se, s, sw, w, nw}



A Wall-Following Program



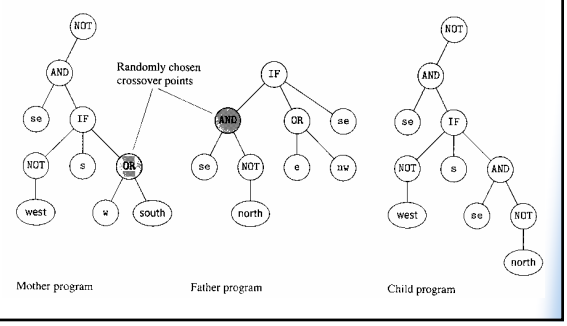

```
(IF (AND (OR (n) (ne)) (NOT (e)))
    (east)
    (IF (AND (OR (e) (se)) (NOT (s)))
        (south)
        (IF (AND (OR (s) (sw)) (NOT (w)))
            (west)
            (north))))
```

Evolving a Wall-Following Robot

- Experimental Setup
  - Population size: 5,000
  - Fitness measure: the number of cells next to the wall that are visited during 60 steps
    - Perfect score (320)
      - One Run (32) × 10 randomly chosen starting points
  - Termination condition: found perfect solution
  - Selection: tournament selection

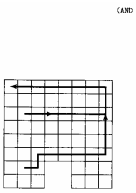
- Creating Next Generation
  - 500 programs (10%) are copied directly into next generation.
    - Tournament selection
      - 7 programs are randomly selected from the population 5,000.
      - The most fit of these 7 programs is chosen.
  - 4,500 programs (90%) are generated by crossover.
    - A mother and a father are each chosen by tournament selection.
    - A randomly chosen subtree from the father replaces a randomly selected subtree from the mother.
  - In this example, mutation was not used.

Two Parents Programs and Their Child



Generation 0

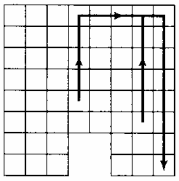
- the most fit program (fitness = 92)
  - Starting in any cell, this program moves east until it reaches a cell next to the wall; then it moves north until it can move east again or it moves west and gets trapped in the upper-left cell.



```
(AND (NOT (NOT (IF (IF (NOT (nw))
                       (IF (e)(north) (east))
                       (IF (west)(0) (south))
                   (OR (IF (nw)(ne)(w))
                       (NOT (sw)))
                   (NOT (NOT (north)))))))
   (IF (OR (NOT (AND (IF (sw)(north)(ne))
                     (AND (south)(1))))
           (OR (OR (NOT (e))
                   (OR (e)(e)))
               (AND (IF (west)(ne)(sw))
                    (IF (1) (e)(e)))))
       (OR (NOT (AND (NOT (ne))(IF (east)(s)(n))))
           (OR (NOT (IF (sw)(east)(s)))
               (AND (IF (w)(sw)(1))
                    (OR (sw)(nw)))))
       (OR (NOT (IF (OR (n)(w))
                    (OR (0)(ne))
                    (OR (1)(east))))
           (OR (AND (OR (1)(ne))
                    (AND (nw)(east)))
               (IF (NOT (west))
                   (AND (west)(east)
                        (IF (1)(north)(e))))))))
```

Generation 2

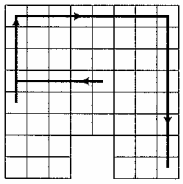
- The most fit program (fitness = 117)
  - Smaller than the best one of generation 0, but it does get stuck in the lower-right corner.



```
(NOT (AND (IF (ne)
              (IF (se)(south)(east))
              (north))
          (NOT (NOT (e)))))
```

Generation 6

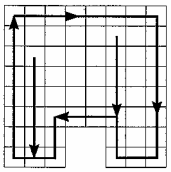
- The most fit program (fitness = 163)
  - Following the wall perfectly but still gets stuck in the bottom-right corner.



```
(IF (AND (NOT (e))
         (IF (e)(s)(nw)))
    (OR (IF (1)(e)(south))
        (IF (north)(east)(nw)))
    (IF (OR (AND (0)(north))
            (AND (e)(IF (e)
                        (IF (se)(south)(east))
                        (north))))
        (AND (e)
             (NOT (IF (s)(sw)(e)))
        (OR (OR (AND (nw)(east))
                (west))
            (nw))))
```

Generation 10

- The most fit program (fitness = 320)
  - Following the wall around clockwise and moves south to the wall if it doesn't start next to it.

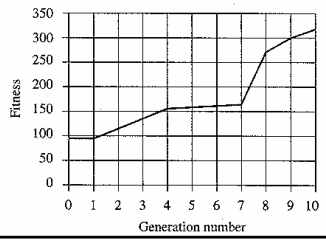


```
(IF (IF (IF (se)(0)(ne))
        (OR (se)(east))
        (IF (OR (AND (e)(0))
                (sw))
            (OR (sw)(0))
            (AND (NOT (NOT (AND (s)(se))))
                 (se))))
    (IF (w)
        (OR (north)
            (NOT (NOT (s))))
        (west))
    (NOT (NOT (NOT (AND (IF (NOT (south))
                            (se)
                            (w))
                        (NOT (n)))))))
```

Fitness Curve

Fitness as a function of generation number

- The progressive (but often small) improvement from generation to generation



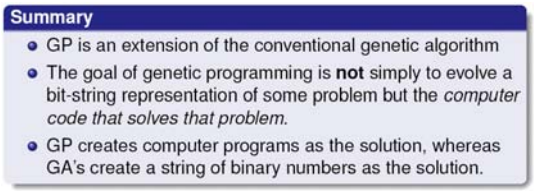
Main advantage of GP

Genetic programming applies the evolutionary approach. However, genetic programming is no longer breeding bit strings that represent coded solutions but **complete computer programs** that solve a particular problem.

Advantages of GP vs GA

- The fundamental difficulty of GAs lies in the problem representation, that is, in the fixed-length coding. A poor representation limits the power of a GA, and even worse, may lead to a false solution.
- A fixed-length coding is rather artificial. As it cannot provide a dynamic variability in length, such a coding often causes considerable redundancy and reduces the efficiency of genetic search. In contrast, genetic programming uses high-level building blocks of variable length. Their size and complexity can change during breeding.

Summary



Applications of EC

- Numerical, Combinatorial Optimization
- System Modeling and Identification
- Planning and Control
- Engineering Design
- Data Mining
- Machine Learning
- Artificial Life

Advantages of EC

- No presumptions about problem space
- Widely applicable
- Low development & application costs
- Easy to incorporate other methods
- Solutions are interpretable (unlike NN)
- Can be run interactively, accommodate user proposed solutions
- Provide many alternative solutions

Disadvantages of EC

- No guarantee for optimal solution within finite time
- Weak theoretical basis
- May need parameter tuning
- Often computationally expensive, i.e. slow