Genetic Programming

Presented by:
Faramarz Safi (Ph.D.)
Faculty of Computer Engineering
Islamic Azad University, Najafabad Branch

Chapter 6

The Challenge

- THE CHALLENGE

"How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?"

– Attributed to Arthur Samuel (1959)

Criterion for Success

- CRITERION FOR SUCCESS

"The aim [is] ... to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence."

– Arthur Samuel (1983)

Main Points

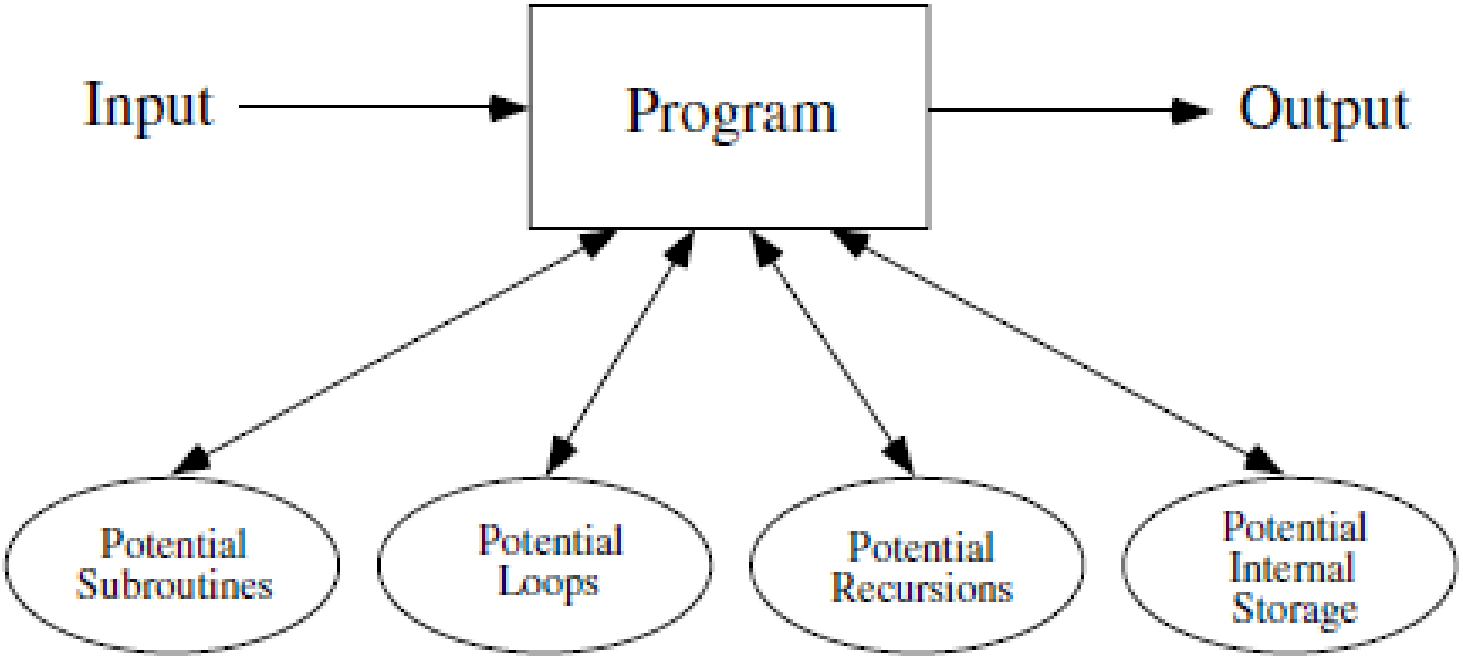
Genetic programming now routinely delivers high-return human-competitive machine intelligence.

Genetic programming is an automated invention machine.
Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology.

SOME (OF THE MANY) REPRESENTATIONS USED TO TRY TO ACHIEVE MACHINE INTELLIGENCE IN THE FIELDS OF ARTIFICIAL INTELLIGENCE (AI) AND MACHINE LEARNING (ML)

- Decision trees
- If-then production rules (e.g., expert systems)
- Horn clauses
- Neural nets (matrices of numerical weights)
- Bayesian networks
- Frames
- Propositional logic
- Binary decision diagrams
- Formal grammars
- Vectors of numerical coefficients for polynomials (adaptive systems)
- Tables of values (reinforcement learning)
- Conceptual clusters
- Concept sets
- Parallel if-then rules (e.g., genetic classifier systems)

A COMPUTER PROGRAM



Introduction

In this chapter we present genetic programming, the youngest member of the evolutionary algorithm family.

Besides the particular representation (using trees as chromosomes), it differs from other EA strands in its application area.

While the EAs discussed so far are typically applied to optimization problems. GP could instead be positioned in machine learning.

This, in fact, is the basis of using evolution for such tasks: models are treated as individuals, and their fitness is the model quality to be maximized.

GP quick overview

- Developed: USA in the 1990's
- Early names: J. Koza
- Typically applied to:
  - machine learning tasks (prediction, classification…)
- Attributed features:
  - competes with neural nets and alike
  - needs huge populations (thousands)
  - slow
- Special:
  - non-linear chromosomes: trees, graphs
  - mutation possible but not necessary (disputed!)

GP technical summary tableau

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Introductory example: credit scoring

As an example we consider a credit scoring problem within a bank that lends money and keeps a track of how its customers pay back their loans.

- This information about the clients can be used to develop a model describing good versus bad customers.
- Later on, this model can be used to predict customers' behavior and thereby assist in evaluating future loan applicants.
- Technically, the classification model is developed based on (historical) data using personal information along with a creditworthiness index (good or bad) of customers.
- The model uses personal data as input, and produces a binary output, standing for the predicted creditworthiness of the corresponding person.
- For instance, the annual salary, the marriage status, and the number of children can be used as input. Table 6.2 shows a small data set.

Introductory example: credit scoring

- Bank wants to distinguish good from bad loan applicants
- Model needed that matches historical data

Customer Id	No. of children	Salary	Marital status	Creditworthiness
Id-1	2	45.000	Married	0
Id-2	0	30.000	Single	1
Id-3	1	40.000	Married	1
Id-4	2	60.000	Divorced	1
...	...	...	...	...
Id-10000	2	50.000	Married	1

Table 6.2. Data for the credit scoring problem

Introductory example: credit scoring

- A possible model:

  IF (No.Childeren = 2) AND (Salary > 80000) THEN good ELSE bad
- In general:

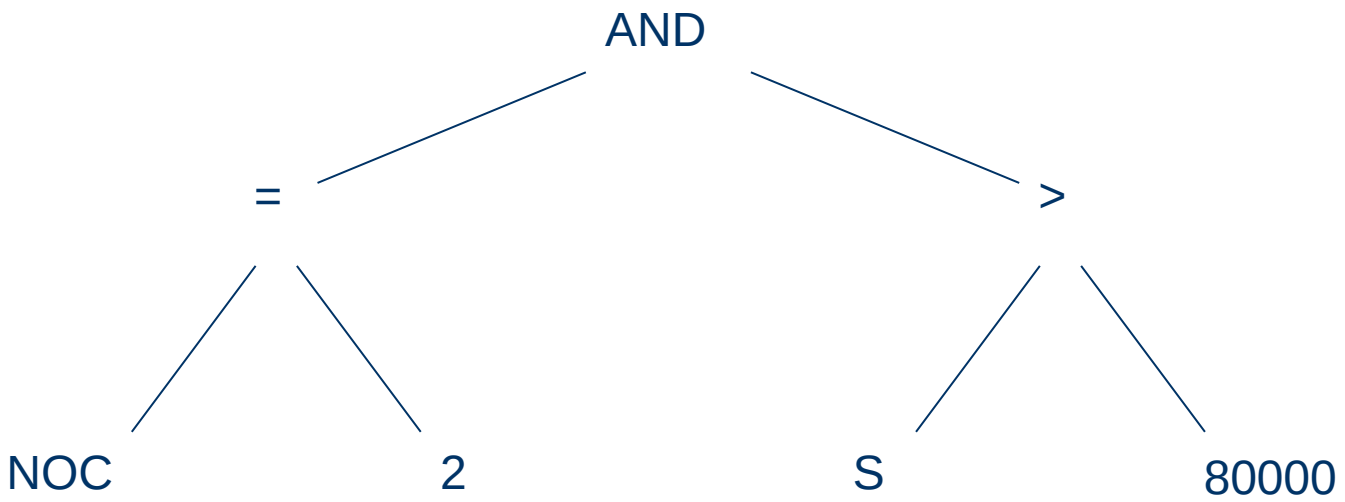
  IF formula THEN good ELSE bad

- Notice that formula is the only unknown in this rule, and all other elements are fixed. Our goal is thus to find the optimal formula that forms an optimal rule classifying a maximum number of known clients correctly.
- At this point we have formulated our problem as a search problem in the space of possible formulas, where the quality of a formula Φ can be defined as the percentage of customers correctly classified by the model IF Φ THEN good ELSE bad.
- In evolutionary terms we have defined the phenotypes (formulas) and the fitness (classification accuracy). In accordance with the typical GP approach we use parse trees as genotypes representing formulas. Fig. 6.1 shows the parse tree of the formula in Eq. (6.1).

Introductory example: credit scoring

This representation differs from those used in GAs or ES in two important aspects:

- The chromosomes are nonlinear structures, while in GAs and ES they are typically linear vectors of the type $V \in D_1 \times ... \times D_n$,where D_i is the domain of v_i.
- The chromosomes can differ in size, measured by the number of nodes of the tree, while in GAs and ES their size, that is, the chromosome length n, is usually fixed.



Introductory example: credit scoring

This new type of chromosomes necessitates new variation operators (crossover and mutation) suitable for trees.

As for selection, notice that it only relies on fitness information and therefore, it is independent from the chromosome structure.

Hence, any selection scheme known in other EAs. e.g. (μ,λ) selection, or fitness proportional with generational replacement, can be simply applied.

Representation

As the introductory example has shown, the general idea in GP is to use parse trees as chromosomes.

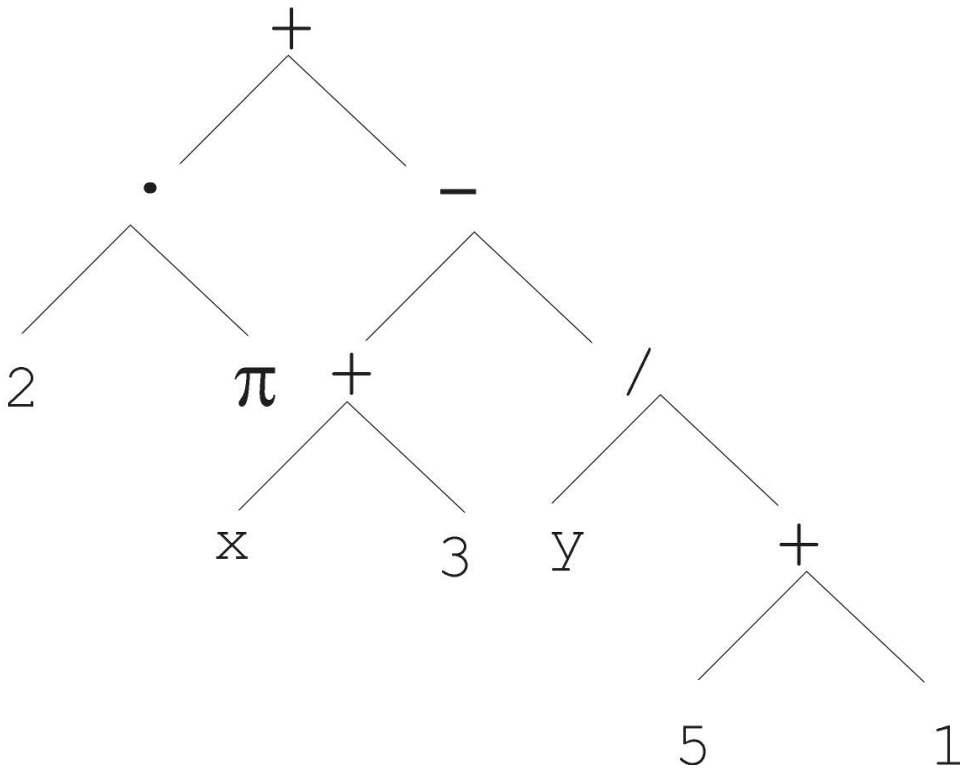
Such parse trees capture expressions in a given formal syntax. Depending on the problem, and the users' perceptions on what the solutions must look like, this can be the syntax of **arithmetic expressions**, **formulas in first-order predicate logic**, or **code written in a programming language**.

Tree based representation

- Trees are a universal form, e.g. consider
- Arithmetic formula $2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$
- Logical formula $(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$
- Program

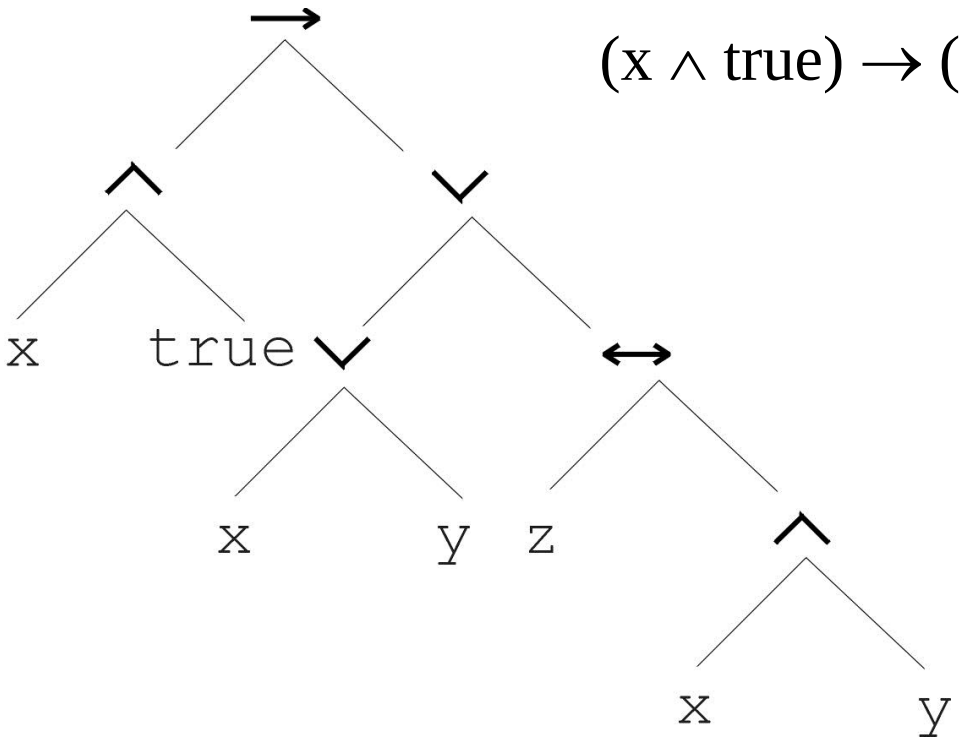
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation



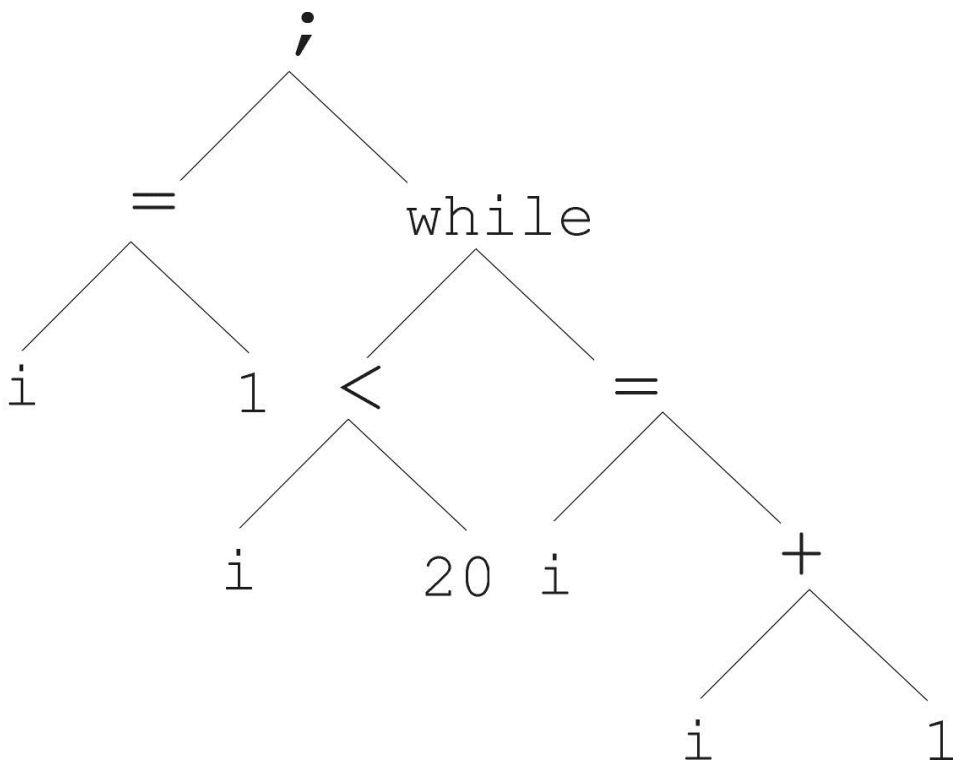
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Tree based representation



$$(x \wedge \text{true}) \rightarrow ((\ x \vee y\ ) \vee (z \leftrightarrow (x \wedge y)))$$

Tree based representation




```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation

- These examples illustrate generally how parse trees can be used and interpreted. Depending on the interpretation GP can be positioned in different ways.
- From a strictly technical point of view, GP is simply a variant of GAs working with a different data structure: the chromosomes are trees. This view disregards application-dependent interpretation issues.
- Nevertheless, such parse trees are often given an interpretation. In particular, they can be envisioned as executable codes, that is, programs.
- The syntax of functional programming, e.g., the language LISP, very closely matches the so-called Polish notation of expressions.
- For instance, the formula in Eq. (6.2) can be rewritten in this Polish notation as **+(-(2, π),-(+(x,3),/(y,+(5,1))))** while the executable LISP code looks like:
  (+ (• 2 π) (- (+ x 3) (/ y (+ 5 1)))).

Tree based representation

- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations).
- Tree shaped chromosomes are non-linear structures.
- In GA, ES, EP the size of the chromosomes is fixed.
- Trees in GP may vary in depth and width.

Tree based representation

GP can be positioned as the "**programming of computers by means of natural selection**", or the "'**automatic evolution of computer programs**".

Technically speaking, the specification of how to represent individuals in GA is analogues to defining the syntax of the **trees**, or equivalently the syntax of the symbolic expressions (**s-expressions**).

This is commonly done by defining a **function set** and a **terminal set**. Elements of the **terminal set are allowed as leaves**, while symbols from the **function set are internal nodes**.

Function set	$\{+, -, \cdot, /\}$
Terminal set	$\mathbb{R} \cup \{x, y\}$

Table 6.3. Function and terminal set that allow the expression in Eq. (6.2) as syntactically correct

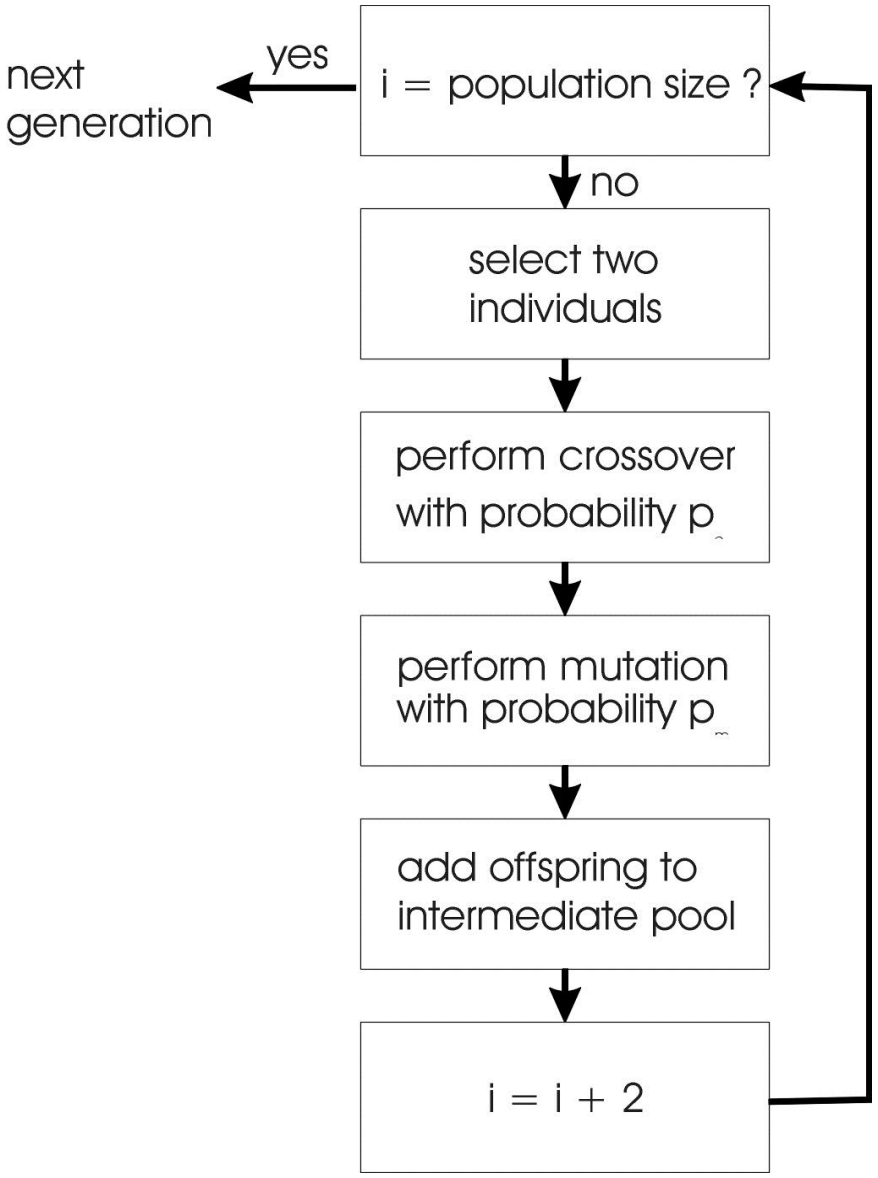
Tree based representation-summary

- Symbolic expressions can be defined by
  - Terminal set T
  - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
  1. Every $t \in T$ is a correct expression
  2. $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, $arity(f)=n$ and $e_1, \ldots, e_n$ are correct expressions
  3. There are no other forms of correct expressions
- In general, expressions in GP are not of any specific type (**closure property**: any $f \in F$ can take any $g \in F$ as argument)

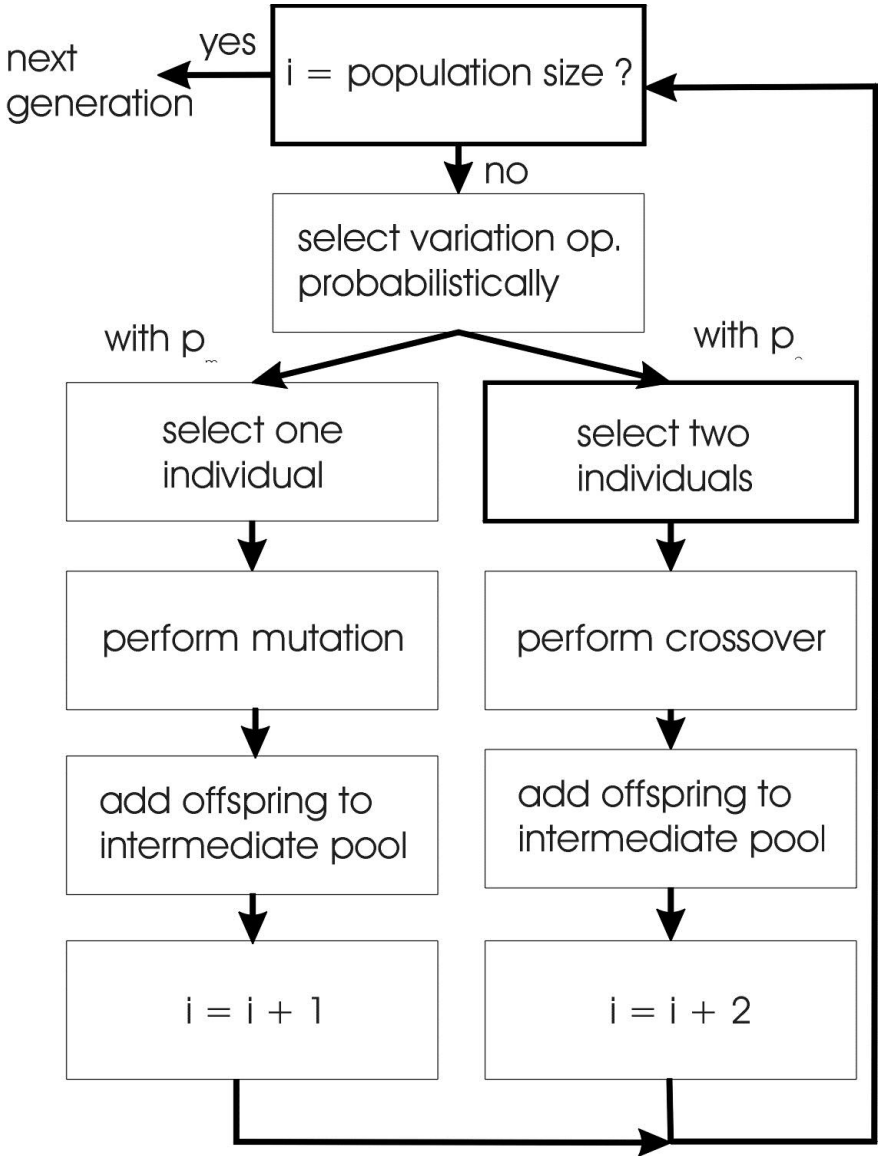
Offspring creation scheme

Compare

- GA and ES scheme using crossover (recombination) AND mutation sequentially (be it probabilistically)
- GP scheme using crossover OR mutation (chosen probabilistically) in one step.



GA flowchart



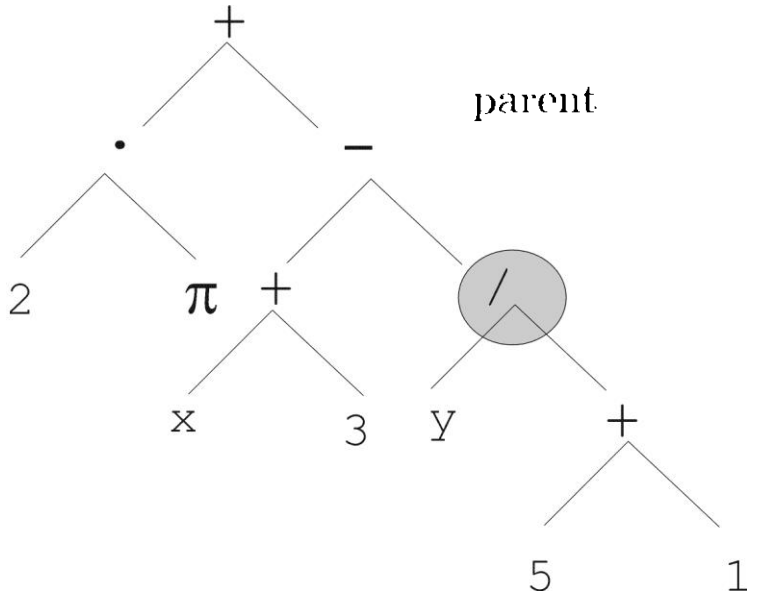
GP flowchart

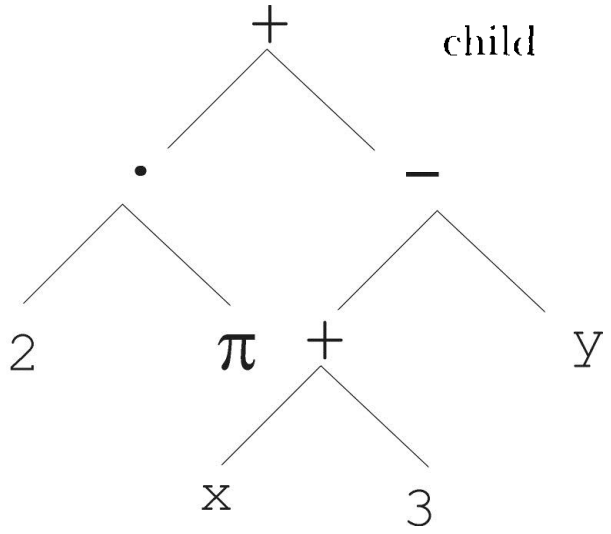
Mutation

- In theory, the task of mutation in GP is the same as in all other EC branches, that is, creating a new individual from an old one through some small random variation.
- The most common implementation works by replacing the subtree starting at a randomly selected node by a randomly generated tree.
- The newly created tree is usually generated the same way as in the initial population.

Mutation

- Fig. 6.5 illustrates how the parse tree1 belonging to Eq. (6.2) (left) is mutated into a parse tree standing for 2 *π + ((x+3)-y).
- Note that the size (tree depth) of the child can exceed that of the parent tree.
- Most common mutation: replace randomly chosen sub tree by randomly generated tree





Mutation cont'd

- Mutation has two parameters:
  - Probability p_m to choose mutation vs. recombination
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (5%) (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

Recombination

Recombination in GP creates offspring by swapping genetic material among the selected parents. In technical terms, it is a binary operator creating two child trees from two parent trees.

The most common implementation is subtree crossover, which works by interchanging the subtrees starting at two randomly selected nodes in the given parents.

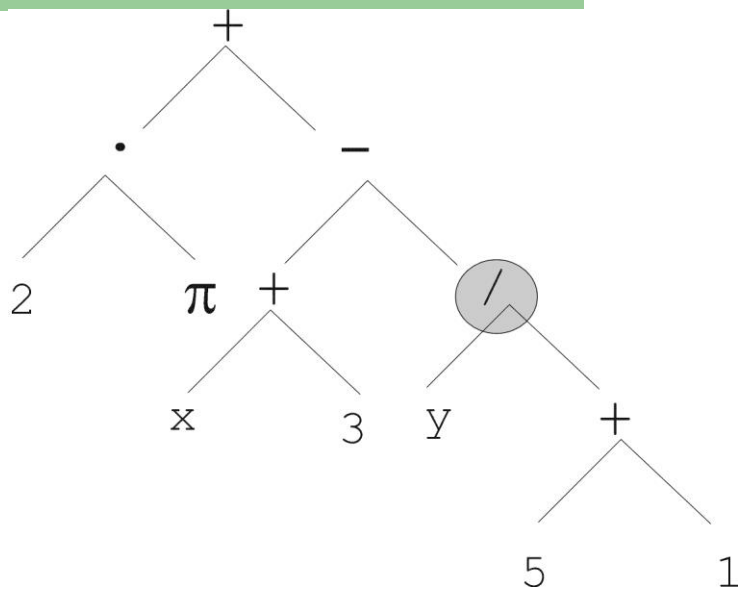
Note that the size (tree depth) of the children can exceed that of the parent trees. In this, recombination within GP differs from recombination in other EC dialects.

Most common recombination (subtree crossover): exchange two randomly chosen subtrees among the parents

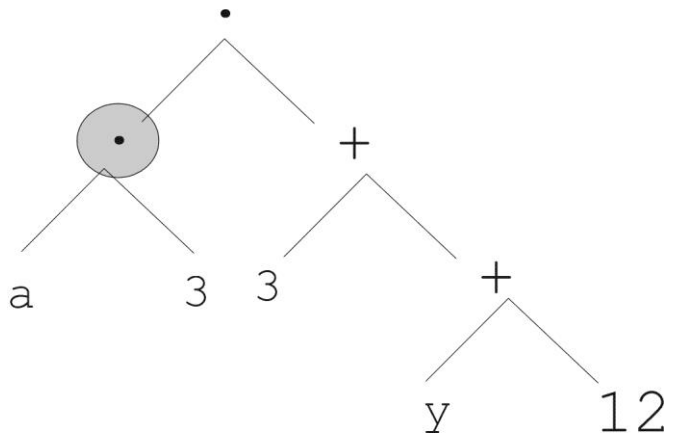
Recombination has two parameters:

- Probability p_c to choose recombination vs. mutation
- Probability to chose an internal point within each parent as crossover point

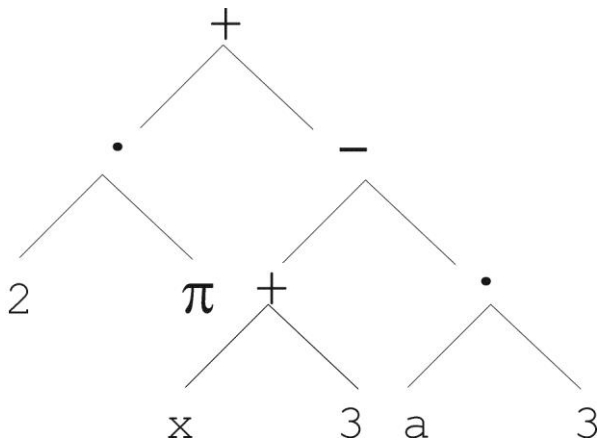
The size of offspring can exceed that of the parents



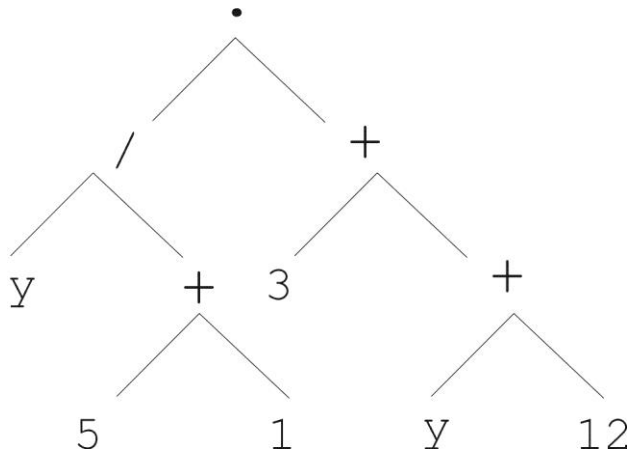
Parent 1



Parent 2



Child 1



Child 2

Parent Selection

GP typically uses **fitness proportionate selection**; however, because of the large population sizes frequently used (population sizes of several thousands are not unusual), a method called **over-selection** is often used for population sizes of 1000 and above.

In this method, the population is first ranked by fitness and then divided into two groups, one containing the top x% and the other containing the other (100-x)%. When parents are selected, 80% of the selection operations come from the first group, and the other 20% from the second group. The values of x used are found empirically by "rule of thumb", and depend on the population size (Table 6.4).

Parent Selection-summary

- Parent selection typically fitness proportionate
- Over-selection in very large populations
  - rank population by fitness and divide it into two groups:
  - group 1: best x% of population, group 2 other (100-x)%
  - 80% of selection operations chooses from group 1, 20% from group 2
  - for pop. size = 1000, 2000, 4000, 8000 x = 32%, 16%, 8%, 4%
  - motivation: to increase efficiency, %'s come from rule of thumb as above
- Survivor selection:
  - Typical: generational scheme (thus none)
  - Recently steady-state is becoming popular for its elitism

Survivor Selection

- Traditional GP typically uses a generational strategy with no elitism. i.e. the number of offspring created is the same as the population size, and all individuals have a life span of one generation. This is, of course, not a technical necessity; rather it is a convention.
- Banzhaf et al. in their book-1998, gave equal treatment to **generational and steady-state** GP, and the latest trend appears to be to use a **steady-state** scheme.
- This is mainly motivated by the need for elitism caused by the destructive effects of crossover.

Initialisation

The most common method of initialization for GP is the so-called **ramped half-and-half** method as follows:

- Maximum initial depth of trees D_{max} is set, and then each member of the initial population is created from the sets of functions F and terminals T using one of the two methods below with equal probability:
- Full method (each branch has depth = D_{max}):
  - nodes at depth $d < D_{max}$ randomly chosen from function set F
  - nodes at depth $d = D_{max}$ randomly chosen from terminal set T
- Grow method (each branch has depth $\leq D_{max}$):
  - nodes at depth $d < D_{max}$ randomly chosen from $F \cup T$
  - nodes at depth $d = D_{max}$ randomly chosen from T

Bloat in Genetic Programming

- A special effect of varying chromosome sizes in GP is that these tend to grow during a GP run.
- That is, without appropriate countermeasures the average tree size is growing during the search process. This phenomenon is known as bloat (sometimes called the "survival of the fattest").
- There are many studies devoted to understanding why bloat occurs and to proposing countermeasures. Appropriate countermeasures to reduce, or even eliminate, tree growth range from elementary to highly sophisticated.
- Probably the simplest way to prevent bloat is to introduce a maximum tree size and forbid a variation operator if the child(ren) resulting from its application would exceed this maximum size.
- In this case, this threshold can be seen as an additional parameter of mutation and recombination in GP.
- Several advanced techniques have also been proposed over the history of GP. But practically the only one that is widely acknowledged is that of **parsimony pressure**. Such a pressure towards parsimony (i.e., being "stingy" or ungenerous) is achieved through **introducing a penalty term in the fitness formula that reduces the fitness of large chromosomes or using multi-objective techniques**.

Problems involving “physical” environments

- An important category of GP applications arises in contexts where executing a given expression, changes the environment, which in turn affects the execution (and therefore the fitness) of the expression.
- The arithmetic examples so far and the symbolic regression application (next example in Sec. 6.11) do not fall into this category.
- They are what can be termed **data fitting problems**, which form the canonical challenge for many machine learning algorithms, GP included.
- This section illustrates another type of problem that we describe as problems involving "physical" environments, and the environment can actually be simulated.
- Together with data fitting, such problems amount to the great majority of GP applications.

Problems involving "physical" environments

- These problems do not have solutions that are simple mappings from the inputs to the outputs: some kind of state information or memory is needed to operate well in these domains.
- As a simple example, consider the **problem of evolving a robot controller** (which is a computer program) for a robot that must walk down a trajectory and collect objects

Function set	glue, if-object-found
Terminal set	pick, move-on

Table 6.5. Function and terminal set for a simple robot controller

Problems involving “physical” environments

- The expressions in the terminal set denote elementary actions that can be performed by the robot.
- Executing pick, results in collecting the object (if any) positioned on the present location of the robot, while move-on causes the robot to shift one position along the trajectory.
- The functions in the function set are both binary:
  - Glue simply concatenates two arguments that is executing glue(x,y) will first perform x and then y.
  - The expression if-object-found(x,y) represents a conditional action based on the state of the environment: if there is an object on the present location of the robot then x is executed, otherwise y.

Problems involving “physical” environments

- This syntax allows if-object-found (pick, move) and if-object-found(move, pick) both are correct expressions (programs).
- The semantics (the environmental effects) tells us that the first one would work well in performing the object collection task, and the second one would not be very effective.
- What distinguishes this kind of GP application is not the internal EA mechanics. Variation, selection, population update, the same way as for all other applications.
- **The difference lies in the fact that a simulator is necessary to calculate fitness by measuring achieved performance in the given environment**.
- In a simple setting, like the above example, one trajectory the robot has to handle. Evaluating the fitness of a given chromosome (where the program is the robot controller) happens by running a simulator that computes the results of the robot's actions as driven by this controller.
- Technically speaking, **this means a fitness evaluation that is computationally expensive**. In a more advanced setting, the fitness evaluations become very expensive, meaning that **one run can take up to months**.

Problems involving "physical" environments- Summary

- Trees for data fitting vs. trees (programs) that are "really" executable
- Execution can change the environment → the calculation of fitness
- Example: robot controller
- Fitness calculations mostly by simulation, ranging from expensive to extremely expensive (in time)
- But evolved controllers are often to very good

Example application: symbolic regression

One of the standard applications of genetic programming is symbolic regression. In the one-dimensional case the problem instance to be solved is defined by a number of pairs $\langle x_i, y_i \rangle \in \mathbb{R} \times \mathbb{R}$ $(i \in \{1, \ldots, n\})$, and the task is to find a function $f : \mathbb{R} \rightarrow \mathbb{R}$ such that $f(x_i) = y_i$ for all $i \in \{1, \ldots, n\}$. It is common to consider the given pairs $\langle x_1, y_i \rangle \in \mathbb{R} \times \mathbb{R}$ as data points and see the whole task as curve fitting. IN other words, we are looking for a function whose plot curve contains each data point.

Example application: symbolic regression

To define a GP algorithm for this problem we start by defining the representation, that is, **the syntax of the s-expressions**, or **parse trees**, used as candidate solutions.

Since the problem is arithmetic, we can use arithmetic operators as functions, for instance { + ,-,.,/ }, but we can also include **power, exp, and elementary geometric functions like sin or cos**.

As for the terminals, we need exactly one variable x and some constants. To keep things simple we allow all elements of IR as constants.

Function set	$\{+, -, \cdot, /, exp, sin, cos\}$
Terminal set	$\mathbb{R} \cup \{x\}$

Table 6.6. Function and terminal set for a symbolic regression problem

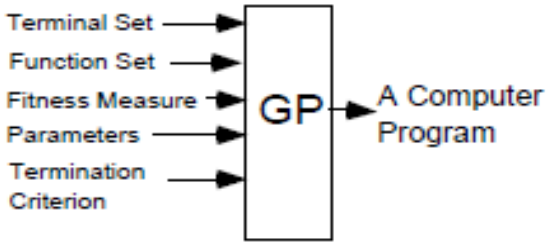
Example application: symbolic regression

- Strictly speaking, we also have to specify the arity of the operators, e.g. sin is unary, and + is binary, but using standard operators as we do this is now self-evident.
- The closure property is not a problem either since all operators are arithmetical here.
- As the following step we define the fitness of an individual *f*, that is, an expression in the above syntax.
- The most natural definition is to base the fitness of *f* on some measure of the fitting error. A suitable error measure is, for instance, the well-known sum of squares which obviously should be minimized:

$$err(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

FIVE MAJOR PREPARATORY STEPS FOR GP

- Determining the set of terminals
- Determining the set of functions
- Determining the fitness measure
- Determining the parameters for the run
  - population size
  - number of generations
  - minor parameters
- Determining the method for designating a result and the criterion for terminating a run



Example application: symbolic regression-summary

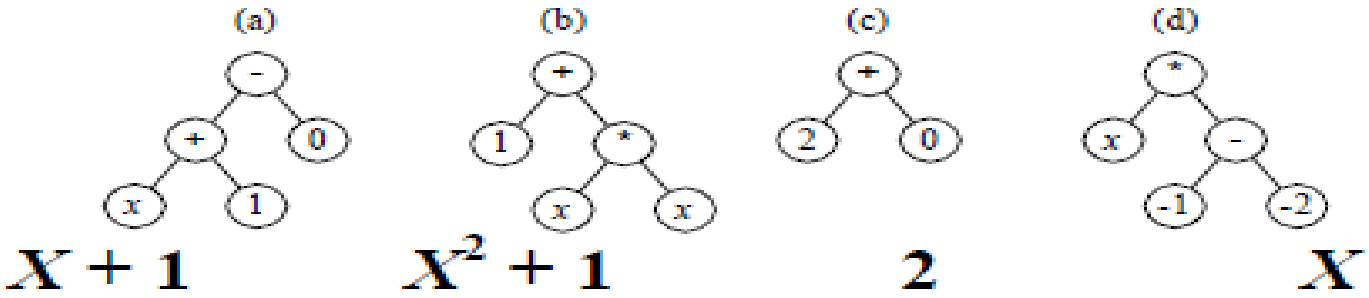
- Given some points in $\mathbf{R}^2$, $(x_1, y_1), \ldots, (x_n, y_n)$
- Find function f(x) s.t. $\forall i = 1, \ldots, n : f(x_i) = y_i$
- Possible GP solution:
  - Representation by F = {+, -, /, sin, cos}, T = $\mathbf{R} \cup \{x\}$
  - Fitness is the error $err(f) = \sum_{i=1}^{n} (f(x_i) - y_i)^2$
  - All operators standard
  - pop.size = 1000, ramped half-half initialisation
  - Termination: n “hits” or 50000 fitness evaluations reached (where “hit” is if $| f(x_i) - y_i | < 0.0001$)

TABLEAU FOR SYMBOLIC REGRESSION OF QUADRATIC POLYNOMIAL $X^2 + X + 1$

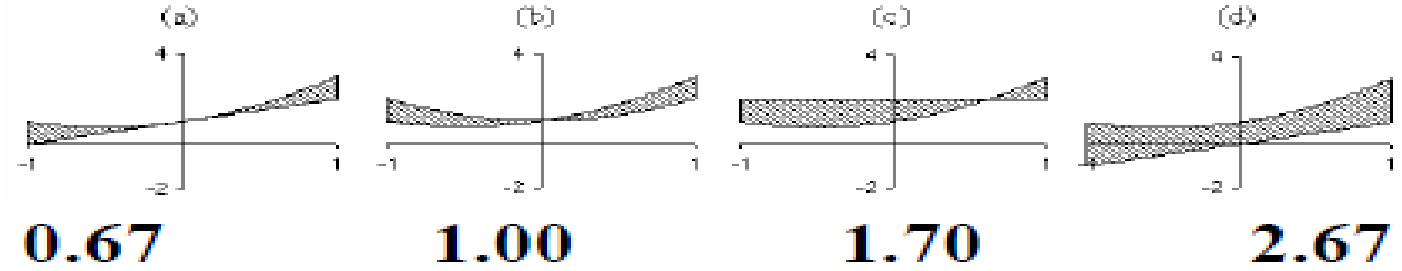
	Objective:	**Find a computer program with one input (independent variable x), whose output equals the value of the quadratic polynomial $x^2 + x + 1$ in range from -1 to +1.**
1	**Terminal set:**	**T = {X}**
2	**Function set:**	**F = {+, -, *, %}** **NOTE: The protected division function % returns a value of 1 when division by 0 is attempted (including 0 divided by 0)**
3	**Fitness:**	**The sum of the absolute value of the differences (errors), computed (in some way) over values of the independent variable x from –1.0 to +1.0, between the program's output and the target quadratic polynomial $x^2 + x + 1$.**
4	**Parameters:**	**Population size $M = 4$.**
5	**Termination:**	**An individual emerges whose sum of absolute errors is less than 0.1**

SYMBOLIC REGRESSION OF QUADRATIC POLYNOMIAL $X^2 + X + 1$

- **INITIAL POPULATION OF FOUR RANDOMLY CREATED INDIVIDUALS OF GENERATION 0**



FITNESS



SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$ (WITH 21 FITNESS CASES)

Independent variable X (Input)	Dependent Variable Y (Output)
-1.0	0.0000
-0.9	-0.1629
-0.8	-0.2624
-0.7	-0.3129
-0.6	-0.3264
-0.5	-0.3125
-0.4	-0.2784
-0.3	-0.2289
-0.2	-0.1664
-0.1	-0.0909
0	0.0
0.1	0.1111
0.2	0.2496
0.3	0.4251
0.4	0.6496
0.5	0.9375
0.6	1.3056
0.7	1.7731
0.8	2.3616
0.9	3.0951
1.0	4.0000

TABLEAU–SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

Objective:	Find a function of one independent variable, in symbolic form, that fits a given sample of 21 (x_i, y_i) data points
Terminal set:	x (the independent variable).
Function set:	+, -, *, %, SIN, COS, EXP, RLOG
Fitness cases:	The given sample of 21 data points (x_i, y_i) where the x_i are in interval [−1,+1].
Raw fitness:	The sum, taken over the 21 fitness cases, of the absolute value of difference between value of the dependent variable produced by the individual program and the target value y_i of the dependent variable.
Standardized fitness:	Equals raw fitness.
Hits:	Number of fitness cases (0 – 21) for which the value of the dependent variable produced by the individual program comes within 0.01 of the target value y_i of the dependent variable.
Wrapper:	None.
Parameters:	Population size, $M = 500$. Maximum number of generations to be run, $G = 51$.
Success Predicate:	An individual program scores 21 hits.

SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

WORST-OF-GENERATION INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 1038

(EXP (- (% X (- X (SIN X)))

(RLOG (RLOG (* X X)))))

Equivalent to

$e^{x/(x-\sin x)} - \log(\log(x*x))$

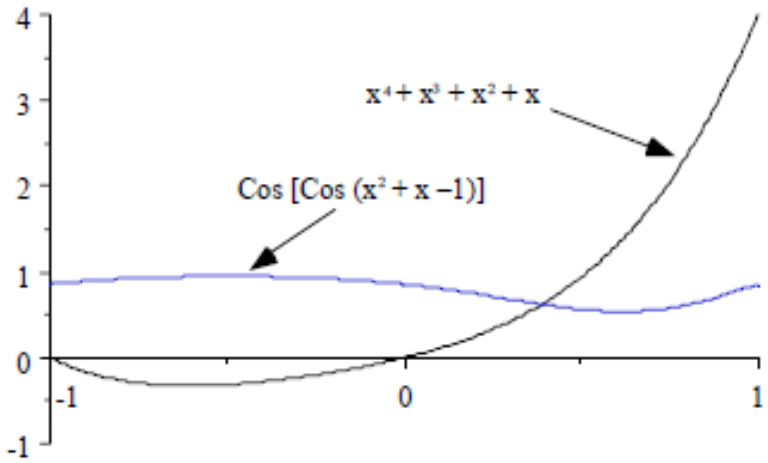
SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

MEDIAN INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 23.67 (AVERGAGE ERROR OF 1.3)

(COS (COS (+ (- (* X X) (% X X)) X)))

Equivalent to

Cos [Cos (x2 + x – 1)]



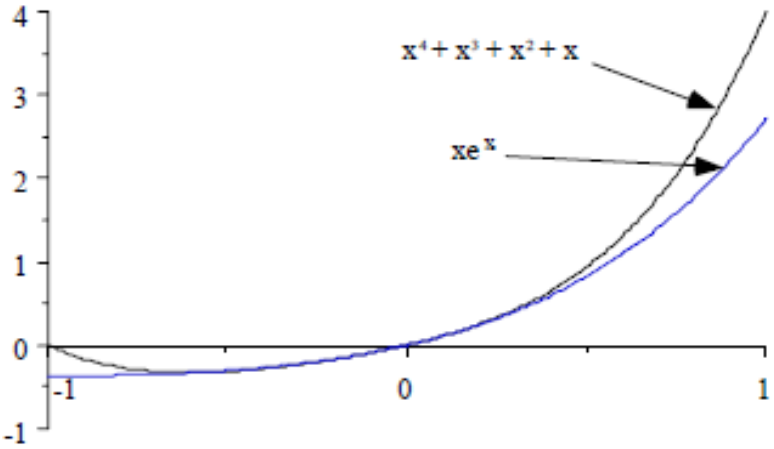
SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

BEST-OF-GENERATION INDIVIDUAL IN GENERATION 0 WITH RAW FITNESS OF 4.47 (AVERGAGE ERROR OF 0.2)

(* X (+ (+ (- (% X X) (% X X)) (SIN (- X X))) (RLOG (EXP (EXP X)))))

Equivalent to

xe^x



SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$ CREATION OF GENERATION 1 FROM GENERATION 0

- In the so-called "generational" model for genetic algorithms, a new population is created that is equal in size to the old population
- 1% mutation (i.e., 5 individuals out of 500)
- 9% reproduction (i.e., 45 individuals)
- 90% crossover (i.e., 225 pairs of parents – yielding 450 offspring)
- All participants in mutation, reproduction, and crossover are chosen from the current population

PROBABILISTICALLY, BASED ON FITNESS

- Anything can happen
- Nothing is guaranteed
- The search is heavily (but not completely) biased toward high-fitness individuals
- The best is not guaranteed to be chosen
- The worst is not necessarily excluded
- Some (but not much) attention is given even to low fitness individuals

SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

BEST-OF-GENERATION INDIVIDUAL IN GENERATION 2 WITH RAW FITNESS OF 2.57 (AVERGAGE ERROR OF 0.1)

(+ (* (* (+ X (* X (* X (% (% X X) (+ X X))))) (+ X (* X X))) X) X)

Equivalent to...

$$x^4 + 1.5x^3 + 0.5x^2 + x$$

SYMBOLIC REGRESSION OF QUARTIC POLYNOMIAL $X^4+X^3+X^2+X$

BEST-OF-RUN INDIVIDUAL IN GENERATION 34 WITH RAW FITNESS OF 0.00 (100%CORRECT)

(+ X (* (+ X (* (* (+ X (- (COS (- X X)) (- X X))) X) X)) X))

Equivalent to

$x^4 + x^3 + x^2 + x$