Evolution strategies

ES quick overview

- Developed: Germany in the 1970's
- Early names: I. Rechenberg, H.-P. Schwefel
- Typically applied to:
  - numerical optimisation
- Attributed features:
  - fast
  - good optimizer for real-valued optimisation
  - relatively much theory
- Special:
  - self-adaptation of (mutation) parameters standard

ES technical summary tableau

Representation	Real-valued vectors
Recombination	Discrete or intermediary
Mutation	Gaussian perturbation
Parent selection	Uniform random
Survivor selection	(μ,λ) or $(\mu+\lambda)$
Specialty	Self-adaptation of mutation step sizes

Introductory example

- Task: minimimise $f : R^n \rightarrow R$
- Algorithm: “two-membered ES” using
  - Vectors from R^n directly as chromosomes
  - Population size 1
  - Only mutation creating one child
  - Greedy selection

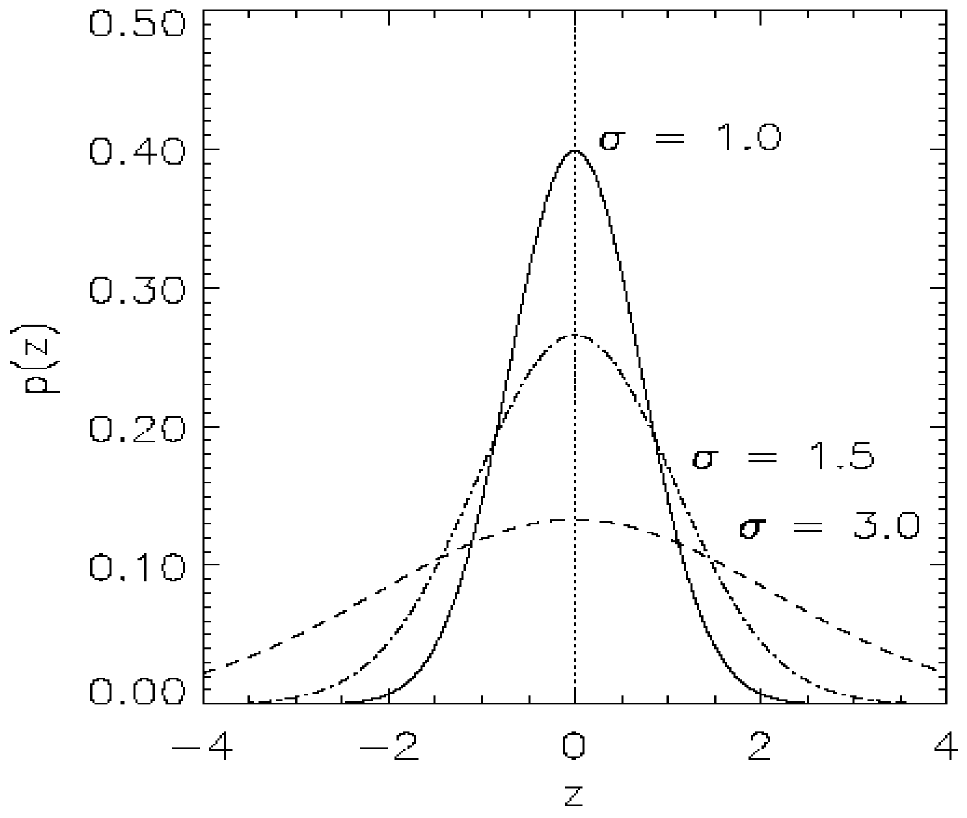
Introductory example: pseudocde

```
Set t = 0
Create initial point $x^t = \langle x_1^t,\ldots,x_n^t \rangle$
REPEAT UNTIL (*TERMIN.COND* satisfied) DO
Draw $z_i$ from a normal distr. for all i = 1,…,n
$y_i^t = x_i^t + z_i$
IF $f(x^t) < f(y^t)$ THEN $x^{t+1} = x^t$
    ELSE $x^{t+1} = y^t$
    FI
    Set t = t+1
OD
```

Introductory example: mutation mechanism

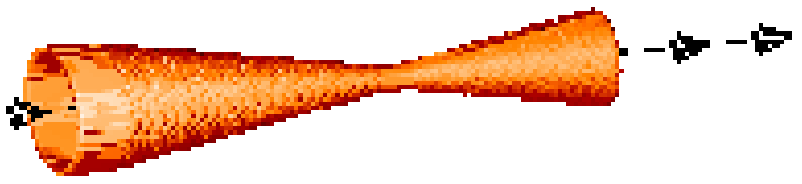
- z values drawn from normal distribution $N(\xi,\sigma)$
  - mean ξ is set to 0
  - variation σ is called mutation step size
- σ is varied on the fly by the "1/5 success rule":
- This rule resets σ after every k iterations by
  - $\sigma = \sigma / c$ if $p_s > 1/5$
  - $\sigma = \sigma \cdot c$ if $p_s < 1/5$
  - $\sigma = \sigma$ if $p_s = 1/5$
- where p_s is the % of successful mutations, $0.8 \leq c \leq 1$

Illustration of normal distribution

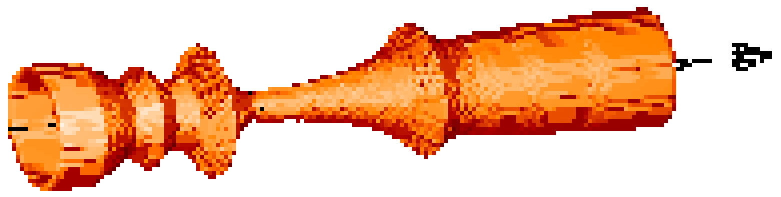


Another historical example: the jet nozzle experiment

Task: to optimize the shape of a jet nozzle
Approach: random mutations to shape + selection



Initial shape



Final shape

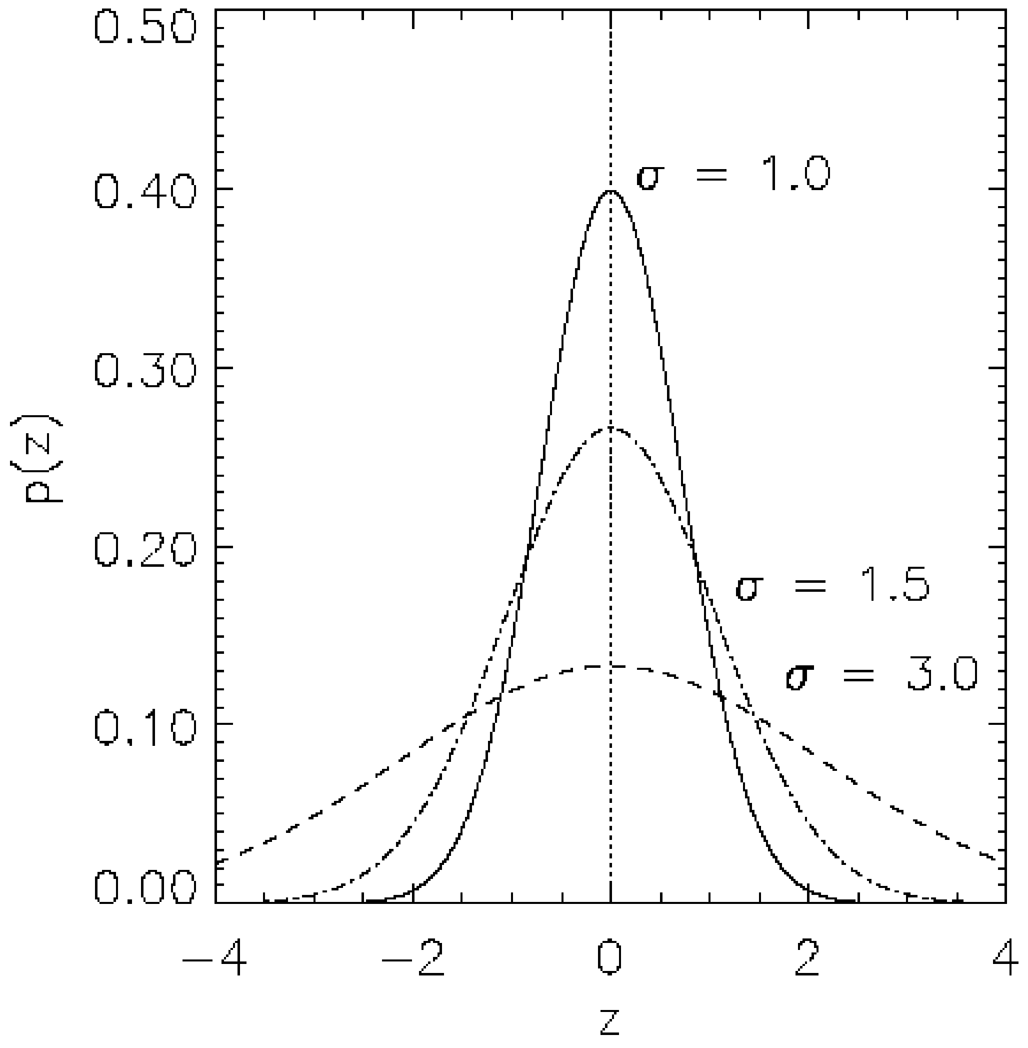
Another historical example: the jet nozzle experiment cont’d

Jet nozzle: the movie

The famous jet nozzle experiment (movie)

Genetic operators: mutations (2)

The one dimensional case



Representation

- Chromosomes consist of three parts:
  - Object variables: $x_1,\dots,x_n$
  - Strategy parameters:
    - Mutation step sizes: $\sigma_1,\dots,\sigma_{n_\sigma}$
    - Rotation angles: $\alpha_1,\dots, \alpha_{n_\alpha}$
- Not every component is always present
- Full size: $\langle x_1,\dots,x_n, \sigma_1,\dots,\sigma_n ,\alpha_1,\dots, \alpha_k \rangle$
- where k = n(n-1)/2 (no. of i,j pairs)

Mutation

- Main mechanism: changing value by adding random noise drawn from normal distribution
- $x'_i = x_i + N(0,\sigma)$
- Key idea:
  - σ is part of the chromosome $\langle x_1,\ldots,x_n, \sigma \rangle$
  - σ is also mutated into σ' (see later how)
- Thus: mutation step size σ is coevolving with the solution x

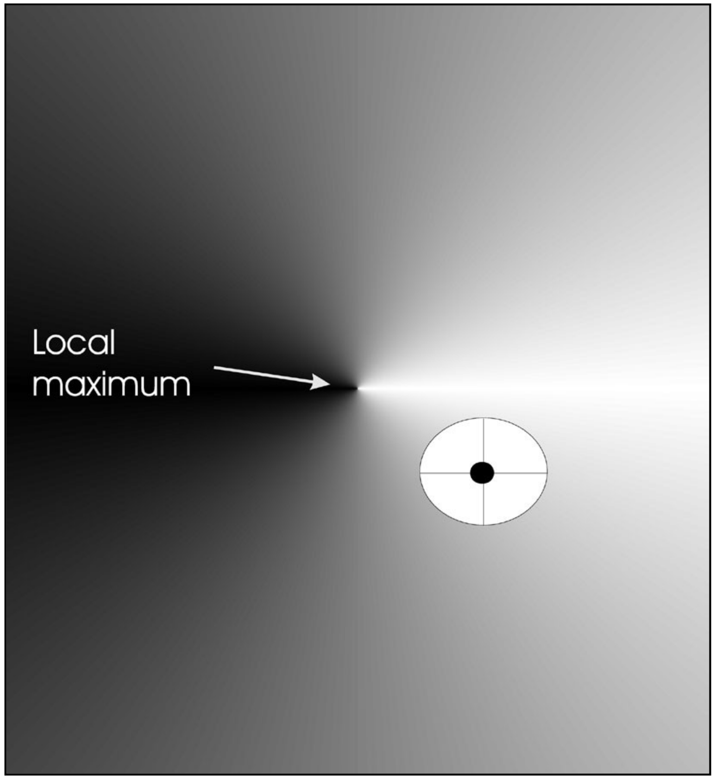
Mutate σ first

- Net mutation effect: $\langle x, \sigma \rangle \rightarrow \langle x', \sigma' \rangle$
- Order is important:
  - first $\sigma \rightarrow \sigma'$ (see later how)
  - then $x \rightarrow x' = x + N(0,\sigma')$
- Rationale: new $\langle x', \sigma' \rangle$ is evaluated twice
  - Primary: x' is good if f(x') is good
  - Secondary: σ' is good if the x' it created is good
- Reversing mutation order this would not work

Mutation case 1: Uncorrelated mutation with one σ

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma \rangle$
- $\sigma' = \sigma \cdot \exp(\tau \cdot N(0,1))$
- $x'_i = x_i + \sigma' \cdot N(0,1)$
- Typically the "learning rate" $\tau \propto 1/ n^{½}$
- And we have a boundary rule $\sigma' < \varepsilon_0 \Rightarrow \sigma' = \varepsilon_0$

Mutants with equal likelihood

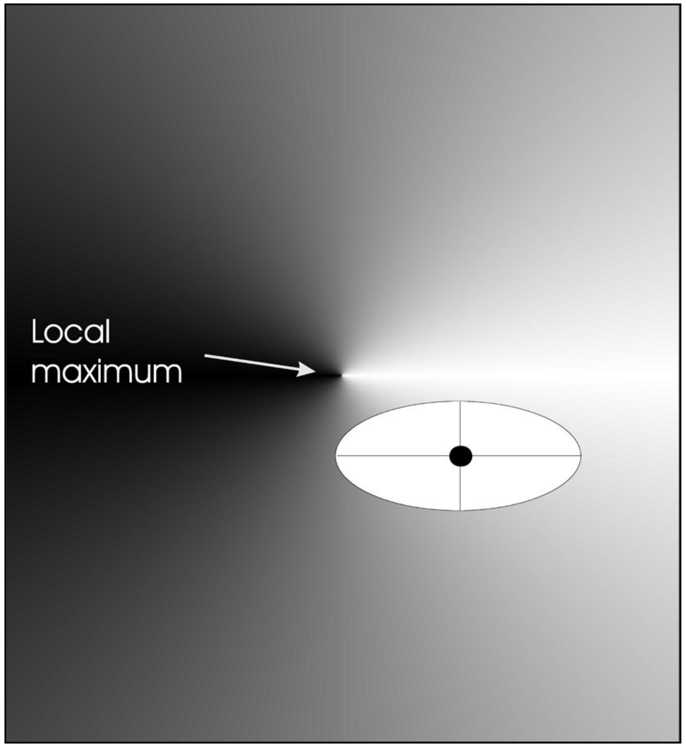


Circle: mutants having the same chance to be created

Mutation case 2: Uncorrelated mutation with n σ's

- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n \rangle$
- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i (0,1))$
- $x'_i = x_i + \sigma'_i \cdot N_i (0,1)$
- Two learning rate parmeters:
  - τ' overall learning rate
  - τ coordinate wise learning rate
- $\tau \propto 1/(2 n)^{½}$ and $\tau \propto 1/(2 n^{½})^{½}$
- And $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$

Mutants with equal likelihood



Ellipse: mutants having the same chance to be created

Mutation case 3: Correlated mutations

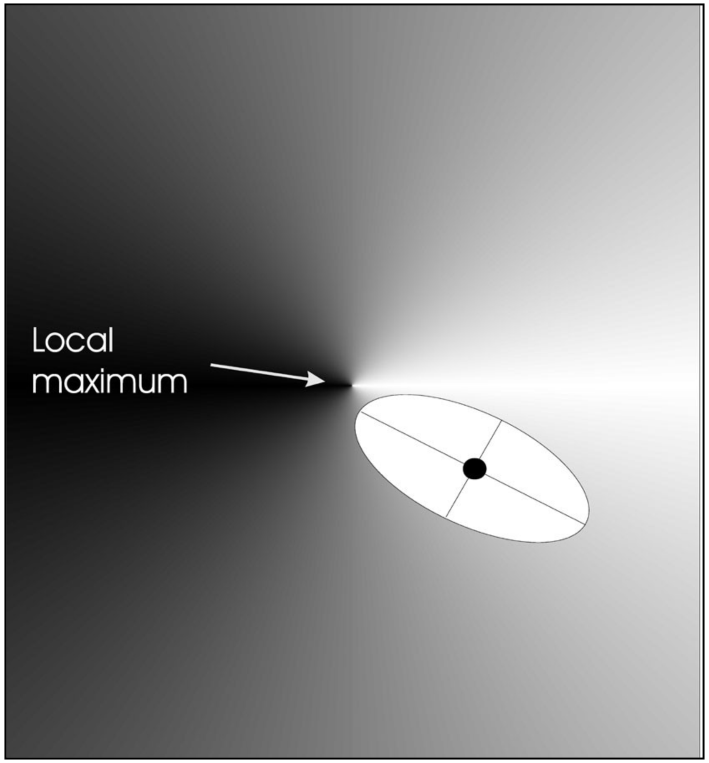
- Chromosomes: $\langle x_1,\ldots,x_n, \sigma_1,\ldots, \sigma_n ,\alpha_1,\ldots, \alpha_k \rangle$
- where $k = n \cdot (n-1)/2$
- and the covariance matrix C is defined as:
  - $c_{ii} = \sigma_i^2$
  - $c_{ij} = 0$ if i and j are not correlated
  - $c_{ij} = ½ \cdot (\sigma_i^2 - \sigma_j^2) \cdot \tan(2 \alpha_{ij})$ if i and j are correlated
- Note the numbering / indices of the α‘s

Correlated mutations cont'd

The mutation mechanism is then:

- $\sigma'_i = \sigma_i \cdot \exp(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1))$
- $\alpha'_j = \alpha_j + \beta \cdot N(0,1)$
- $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{N}(\boldsymbol{0},\boldsymbol{C}')$
  - ***x*** stands for the vector $\langle x_1,\ldots,x_n \rangle$
  - ***C'*** is the covariance matrix ***C*** after mutation of the α values
- $\tau \propto 1/(2n)^{½}$ and $\tau \propto 1/(2n^{½})^{½}$ and $\beta \approx 5°$
- $\sigma_i' < \varepsilon_0 \Rightarrow \sigma_i' = \varepsilon_0$ and
- $|\alpha'_j| > \pi \Rightarrow \alpha'_j = \alpha'_j - 2\pi\, \text{sign}(\alpha'_j)$

Mutants with equal likelihood



Ellipse: mutants having the same chance to be created

Recombination

- Creates one child
- Acts per variable / position by either
  - Averaging parental values, or
  - Selecting one of the parental values
- From two or more parents by either:
  - Using two selected parents to make a child
  - Selecting two parents for each position anew

Names of recombinations

	Two fixed parents	Two parents selected for each i
$z_i = (x_i + y_i)/2$	Local intermediary	Global intermediary
z_i is x_i or y_i chosen randomly	Local discrete	Global discrete

Parent selection

- Parents are selected by uniform random distribution whenever an operator needs one/some
- Thus: ES parent selection is unbiased - every individual has the same probability to be selected
- Note that in ES “parent” means a population (in GA’s parent is a member of the population)

Survivor selection

- Applied after creating λ children from the μ parents by mutation and recombination
- Deterministically chops off the “bad stuff”
- Basis of selection is either:
  - The set of children only: (μ,λ)-selection
  - The set of parents and children: $(\mu+\lambda)$-selection

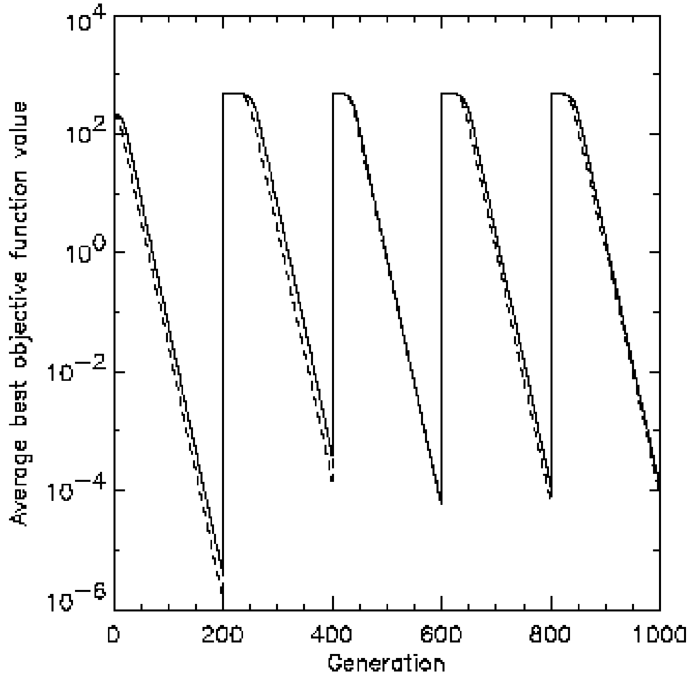
Survivor selection cont'd

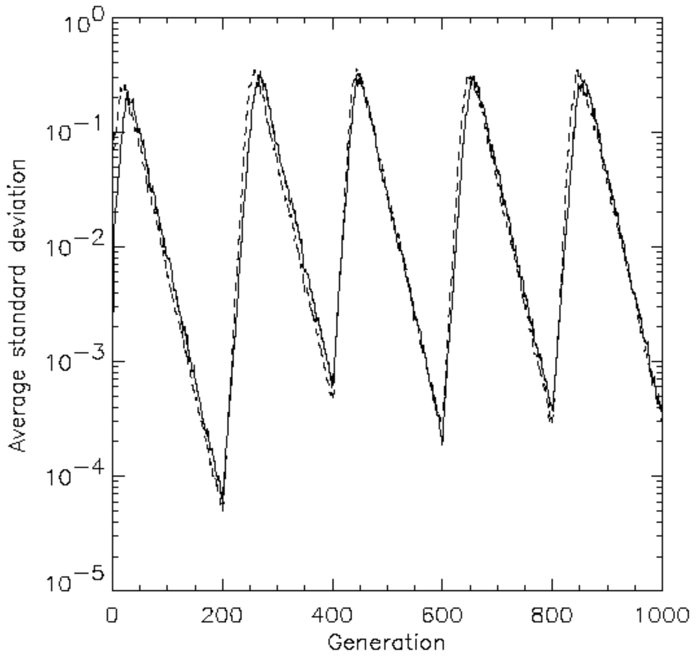
- $(\mu+\lambda)$-selection is an elitist strategy
- (μ,λ)-selection can "forget"
- Often (μ,λ)-selection is preferred for:
  - Better in leaving local optima
  - Better in following moving optima
  - Using the + strategy bad σ values can survive in $\langle x,\sigma\rangle$ too long if their host x is very fit
- Selective pressure in ES is very high ($\lambda \approx 7 \cdot \mu$ is the common setting)

Self-adaptation illustrated

- Given a dynamically changing fitness landscape (optimum location shifted every 200 generations)
- Self-adaptive ES is able to
  - follow the optimum and
  - adjust the mutation step size after every shift !

Self-adaptation illustrated cont'd





Changes in the fitness values (left) and the mutation step sizes (right)

Prerequisites for self-adaptation

- $\mu > 1$ to carry different strategies
- $\lambda > \mu$ to generate offspring surplus
- Not “too” strong selection, e.g., $\lambda \approx 7 \cdot \mu$
- (μ,λ)-selection to get rid of misadapted σ‘s
- Mixing strategy parameters by (intermediary) recombination on them

Example application: the cherry brandy experiment

- Task to create a colour mix yielding a target colour (that of a well known cherry brandy)
- Ingredients: water + red, yellow, blue dye
- Representation: ‹ w, r, y ,b ▯ no self-adaptation!
- Values scaled to give a predefined total volume (30 ml)
- Mutation: lo / med / hi σ values used with equal chance
- Selection: (1,8) strategy

Example application: cherry brandy experiment cont’d

- Fitness: students effectively making the mix and comparing it with target colour
- Termination criterion: student satisfied with mixed colour
- Solution is found mostly within 20 generations
- Accuracy is very good

Example application: the Ackley function (Bäck et al ’93)

- The Ackley function (here used with n =30):

$$f(x) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$$

- Evolution strategy:
  - Representation:
    - $-30 < x_i < 30$ (coincidence of 30’s!)
    - 30 step sizes
  - (30,200) selection
  - Termination : after 200000 fitness evaluations
  - Results: average best solution is $7.48 \cdot 10^{-8}$ (very good)

Genetic Programming

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

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

ID	No of children	Salary	Marital status	OK?
ID-1	2	45000	Married	0
ID-2	0	30000	Single	1
ID-3	1	40000	Divorced	1
…				

Introductory example: credit scoring

- A possible model:

IF (NOC = 2) AND (S > 80000) THEN good ELSE bad

- In general:

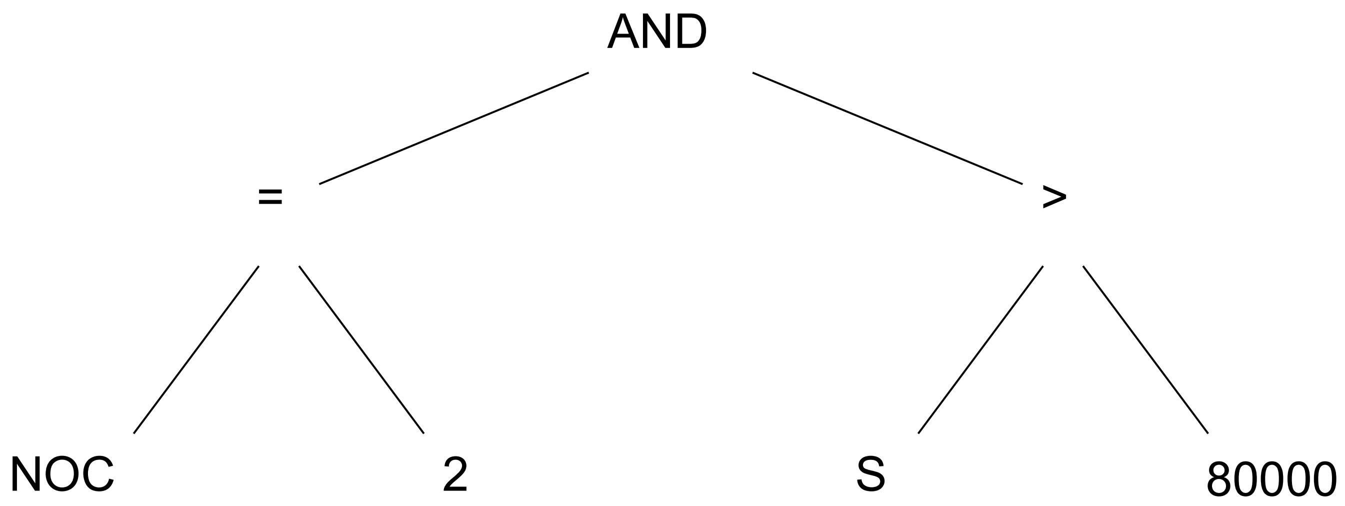
IF formula THEN good ELSE bad

- Only unknown is the right formula, hence
- Our search space (phenotypes) is the set of formulas
- Natural fitness of a formula: percentage of well classified cases of the model it stands for
- Natural representation of formulas (genotypes) is: parse trees

Introductory example: credit scoring

IF (NOC = 2) AND (S > 80000) THEN good ELSE bad

can be represented by the following tree



Tree based representation

- Trees are a universal form, e.g. consider
- Arithmetic formula $$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

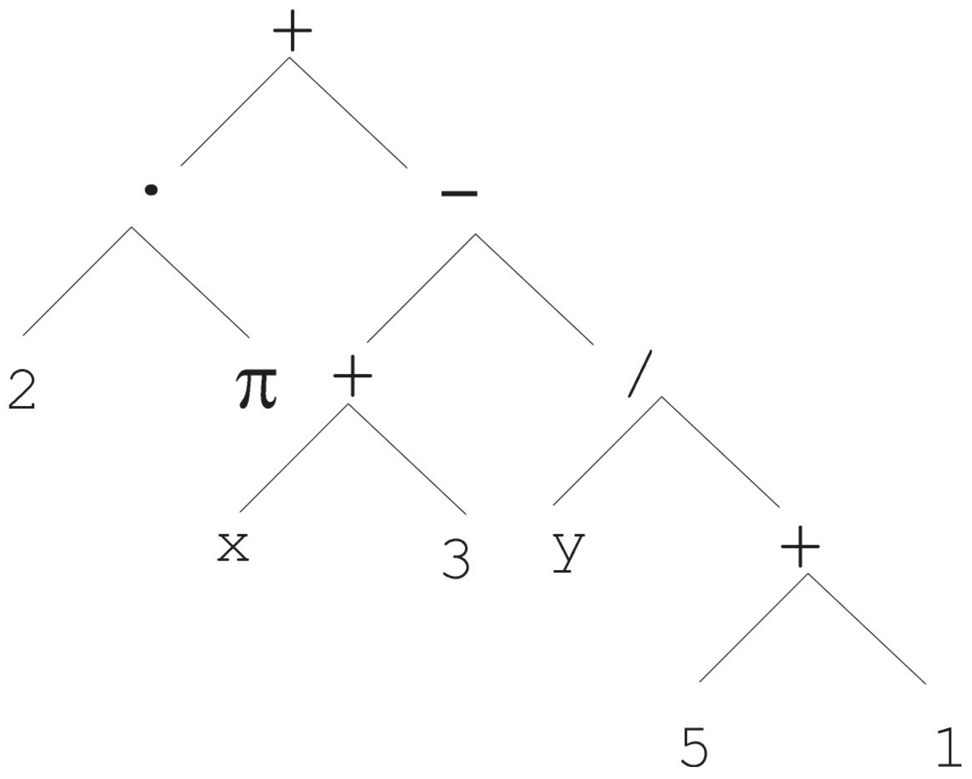
$$(x \wedge \mathrm{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y))$$

- Logical formula

- Program

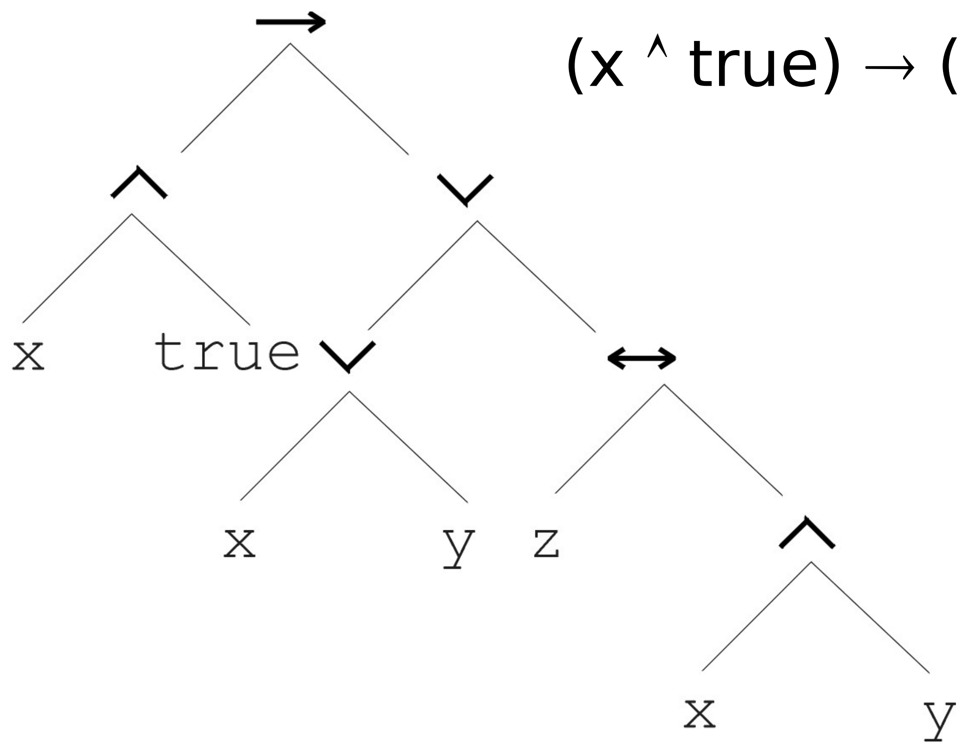
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation



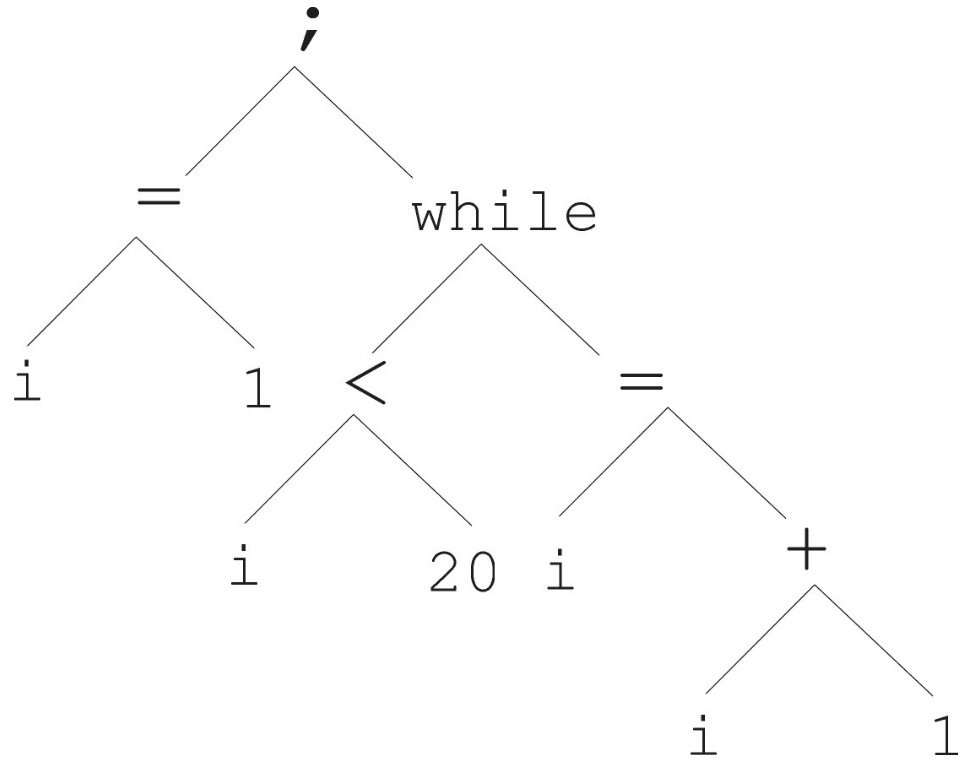
$$2\cdot\pi + \left((x+3) - \frac{y}{5+1}\right)$$

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

- 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

What is Genetic Programming?

- Genetic programming is a model of programming which uses the ideas (and some of the terminology) of biological evolution to handle a complex problem. … Genetic programming can be viewed as an extension of the *genetic algorithm*, a model for testing and selecting the best choice among a set of results, each represented by a string.

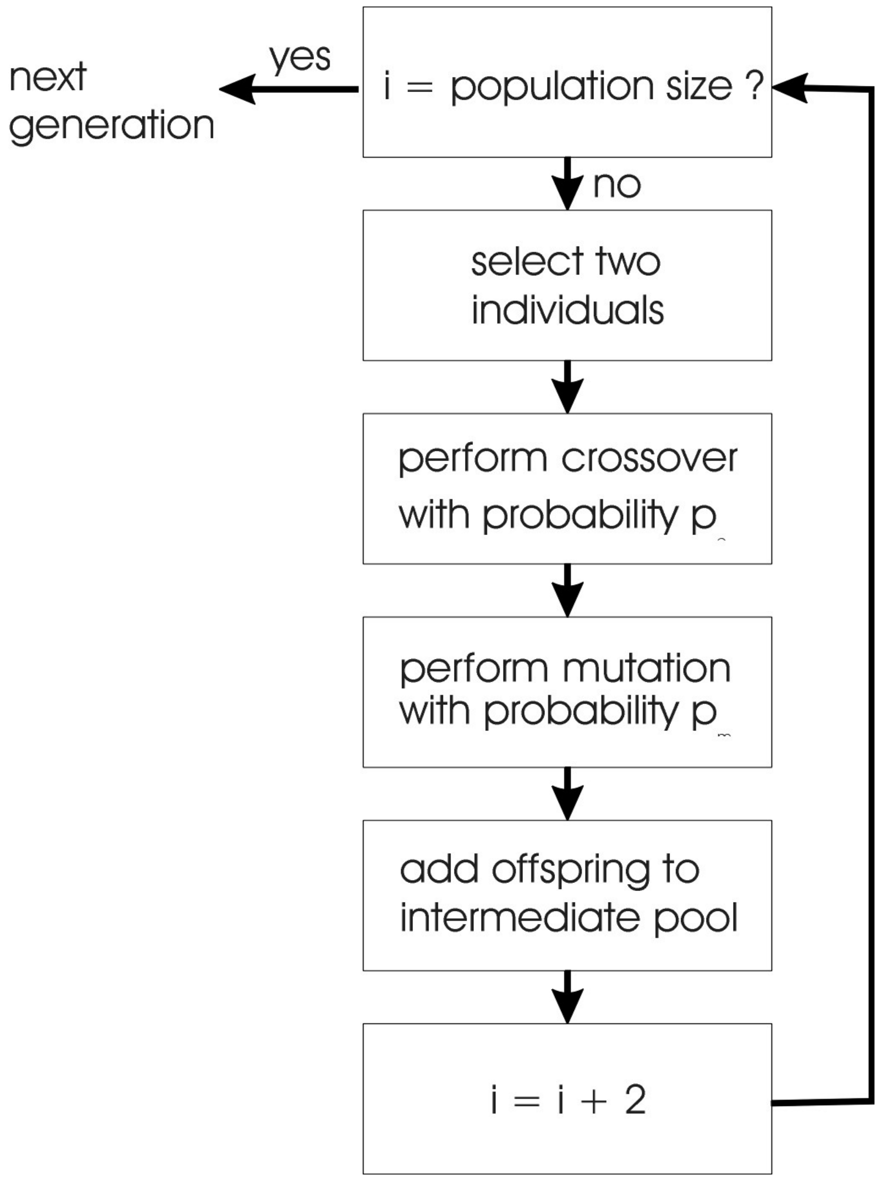
://whatis.techtarget.com/definition/0%2C%2Csid9_gci214639%2C00.html

Outline of the Genetic Algorithm

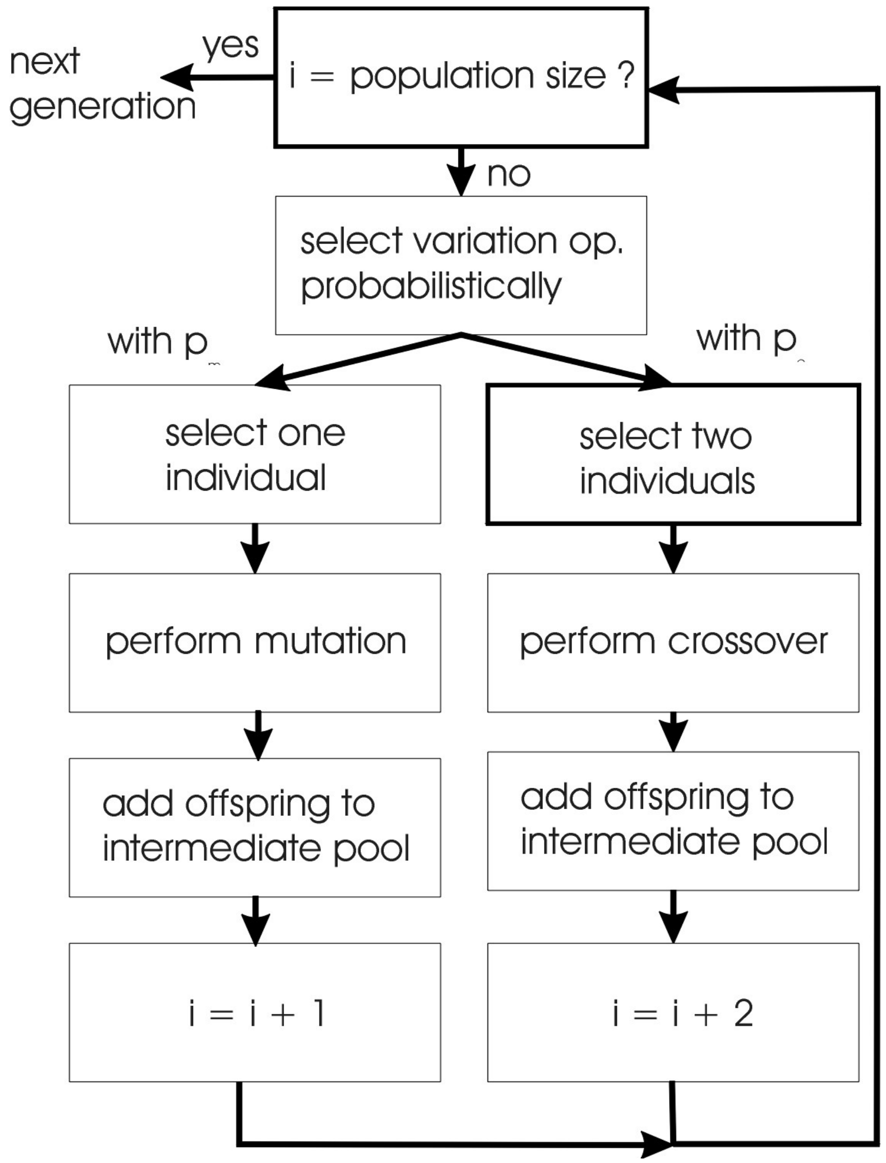
- Randomly generate a set of possible solutions to a problem, representing each as a fixed length character string
- Test each possible solution against the problem using a fitness function to evaluate each solution
- Keep the best solutions, and use them to generate new possible solutions
- Repeat the previous two steps until either an acceptable solution is found, or until the algorithm has iterated through a given number of cycles (generations)

Why Genetic Programming?

- "It is difficult, unnatural, and overly restrictive to attempt to represent hierarchies of dynamically varying size and shape with fixed length character strings." "For many problems in machine learning and artificial intelligence, the most natural representation for a solution is a computer program." [Koza, 1994]
- A parse tree is a good representation of a computer program for Genetic Programming


next generation
yes
i = population size ?
no
select two individuals
perform crossover with probability p_c
perform mutation with probability p_m
add offspring to intermediate pool
i = i + 2

GA flowchart



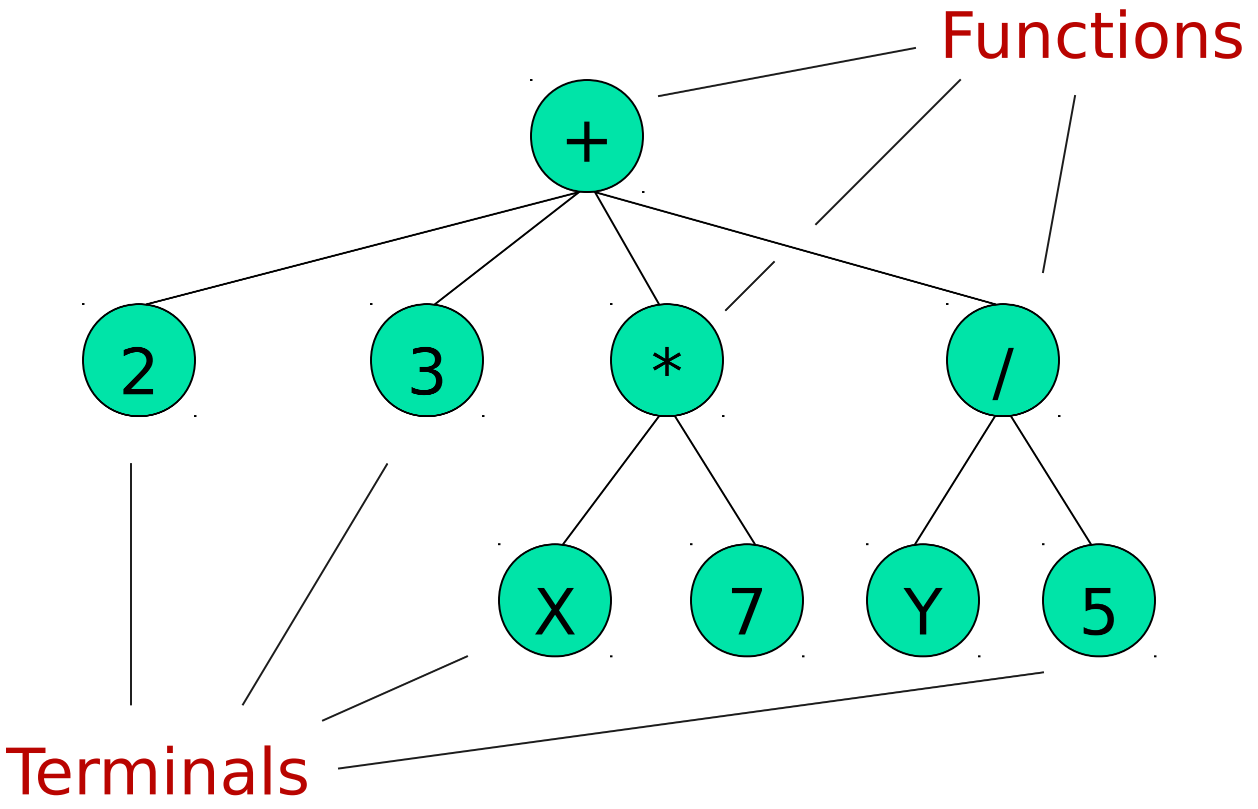
GP flowchart

Tree based representation

- 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 typed (closure property: any $f \in F$ can take any $g \in F$ as argument)

Using Trees To Represent Computer Programs

(+ 2 3 (* X 7) (/ Y 5))



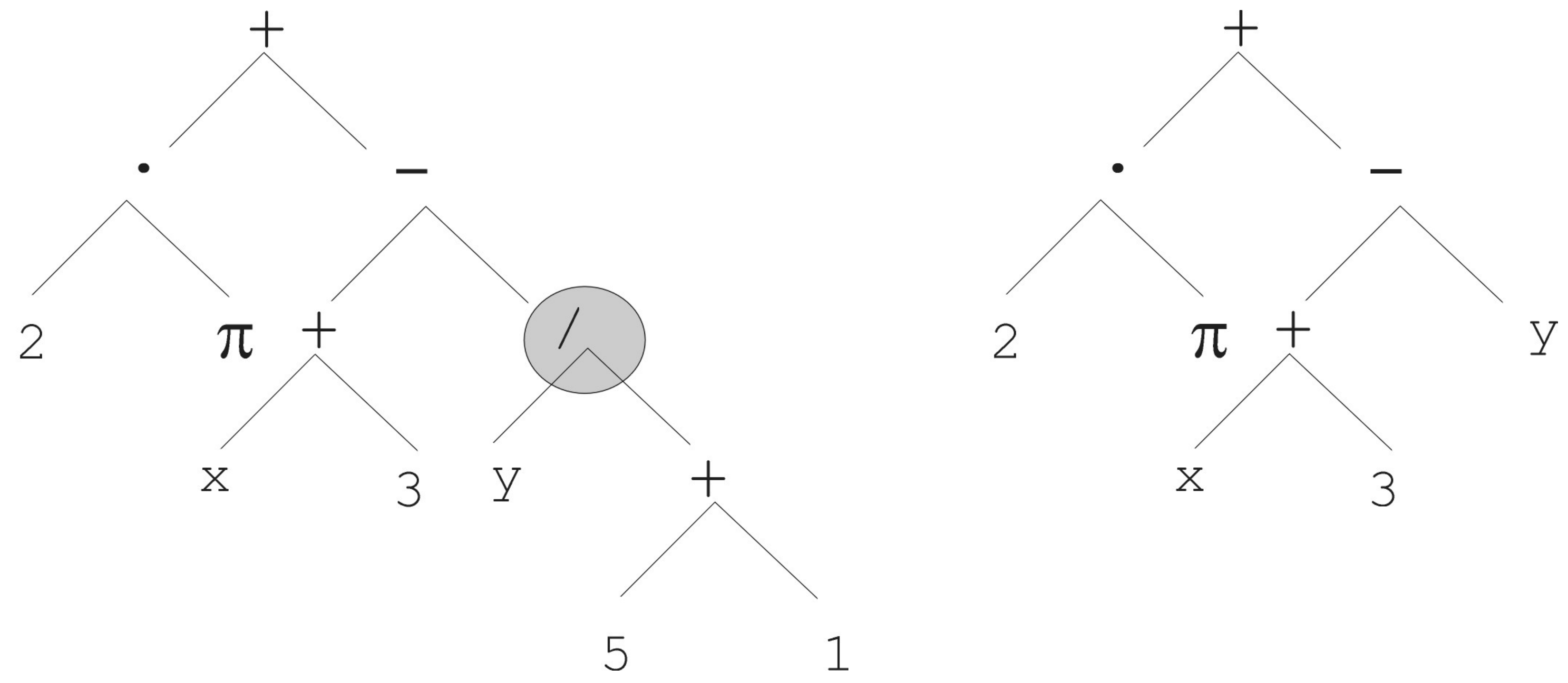
Offspring creation scheme

Compare

- GA scheme using crossover AND mutation sequentially (be it probabilistically)
- GP scheme using crossover OR mutation (chosen probabilistically)

Mutation

- Most common mutation: replace randomly chosen subtree by randomly generated tree

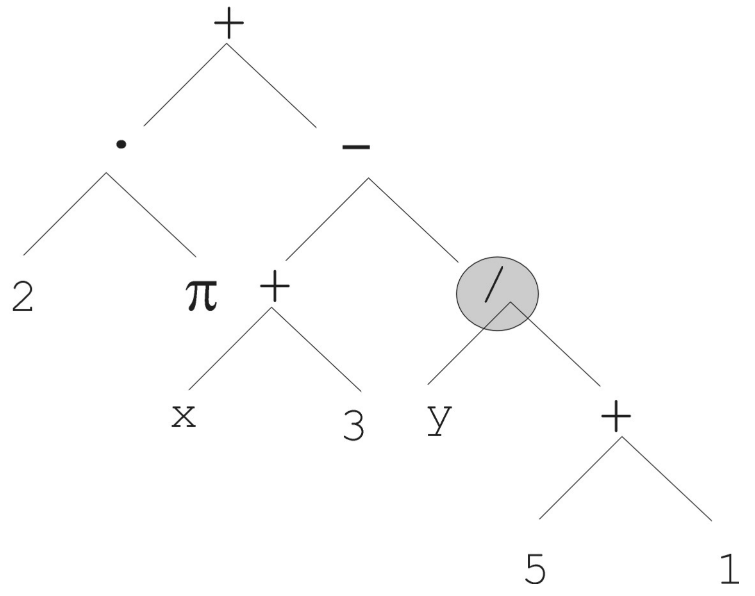


Mutation cont’d

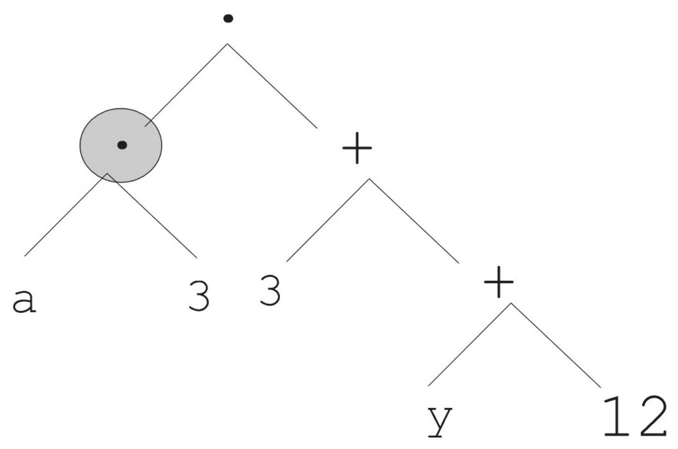
- Mutation has two parameters:
  - Probability p_m to choose mutation vs. recombination
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza’92) or very small, like 0.05 (Banzhaf et al. ’98)
- The size of the child can exceed the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

Recombination

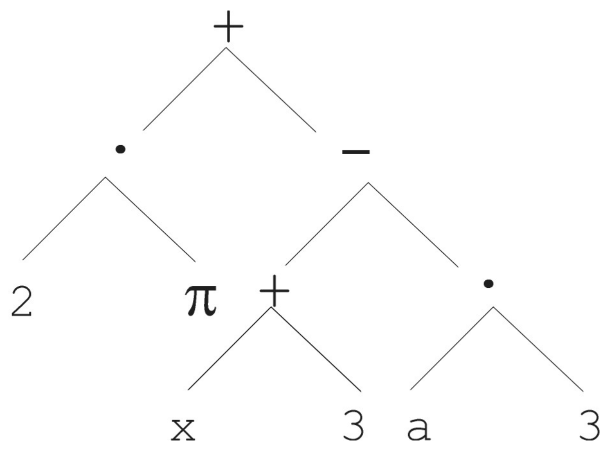
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination vs. mutation
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents



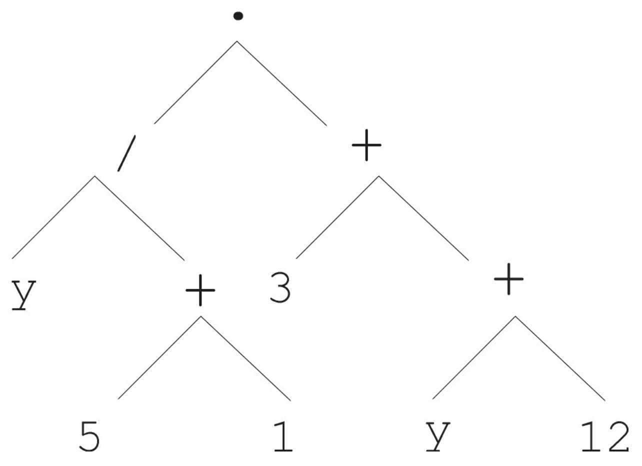
Parent 1



Parent 2



Child 1



Child 2

Selection

- 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

Initialisation

- Maximum initial depth of trees D_{max} is set
- 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
- Common GP initialisation: ramped half-and-half, where grow & full method each deliver half of initial population

Bloat

- Bloat = “survival of the fattest”, i.e., the tree sizes in the population are increasing over time
- Ongoing research and debate about the reasons
- Needs countermeasures, e.g.
  - Prohibiting variation operators that would deliver “too big” children
  - Parsimony pressure: penalty for being oversized

Problems involving “physical” environments

- 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

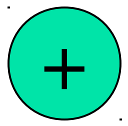
- Given some points in $\mathbf{R}^2$, $(x_1, y_1), \ldots, (x_n, y_n)$
- Find function $f(x)$ s.t. $\forall i = 1, \ldots, n : f(x_i) = y_i$
- Possible GP solution:
  - Representation by $F = \{+, -, /, \sin, \cos\}$, $T = \mathbf{R} \cup \{x\}$
  - Fitness is the error $$err(f) = \sum_{i=1}^{n} (f(x_i)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

Randomly Generating Programs

- Randomly generate a program that takes two arguments and uses basic arithmetic to return an answer
  - Function set = {+, -, *, /}
  - Terminal set = {integers, X, Y}
- Randomly select either a function or a terminal to represent our program
- If a function was selected, recursively generate random programs to act as arguments

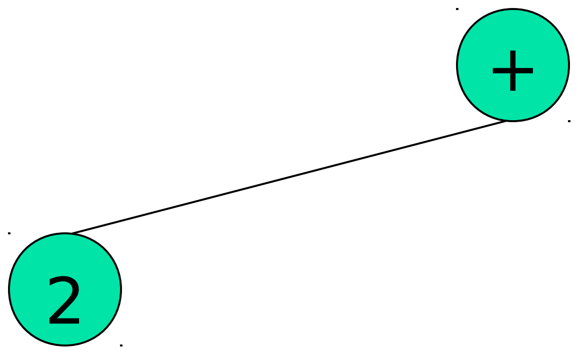
Randomly Generating Programs

(+ ...)



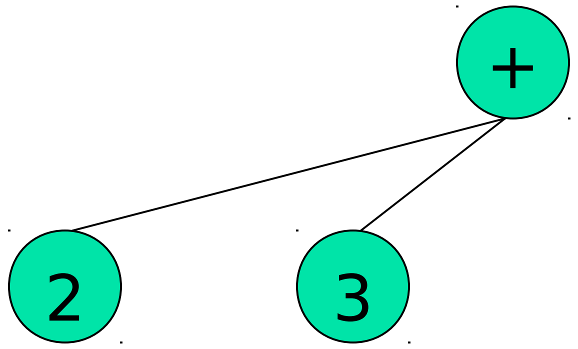
Randomly Generating Programs

(+ 2 ...)



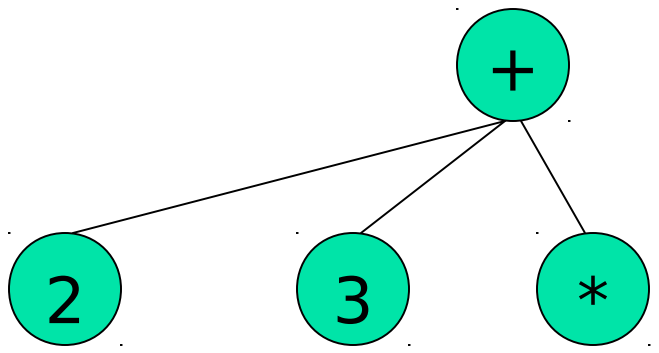
Randomly Generating Programs

(+ 2 3 ...)



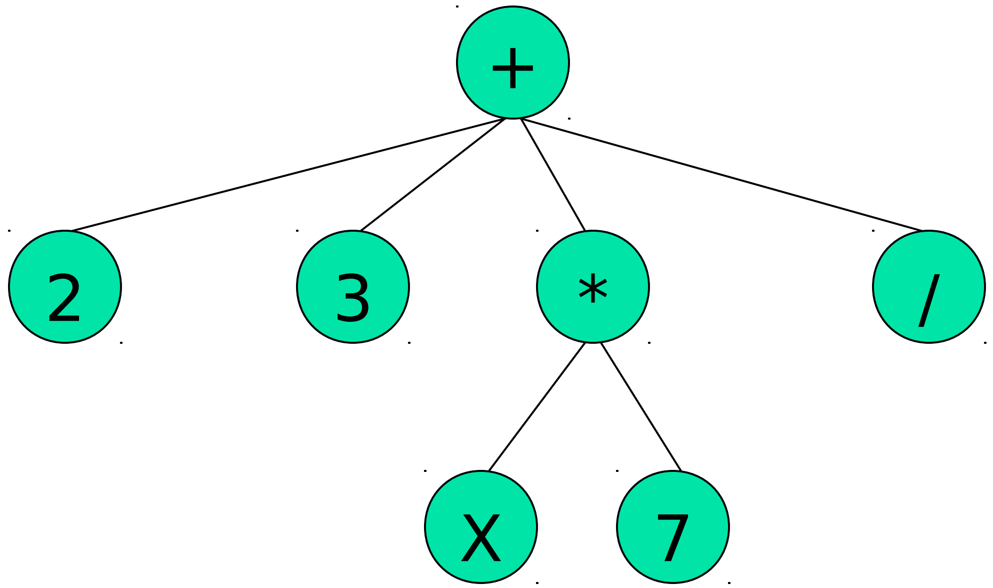
Randomly Generating Programs

(+ 2 3 (* ...) ...)



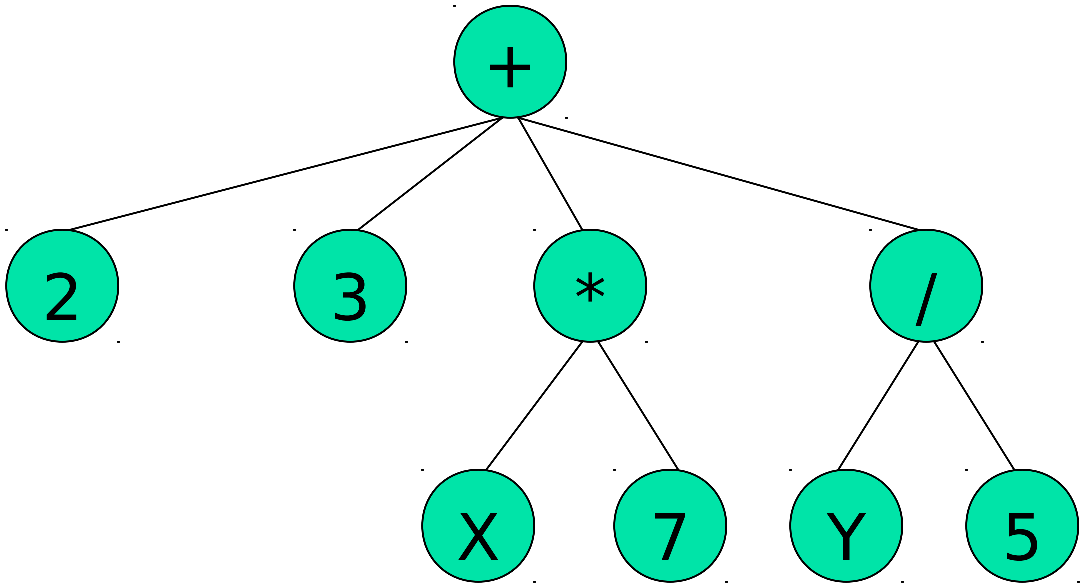
Randomly Generating Programs

(+ 2 3 (* X 7) (/ ...))



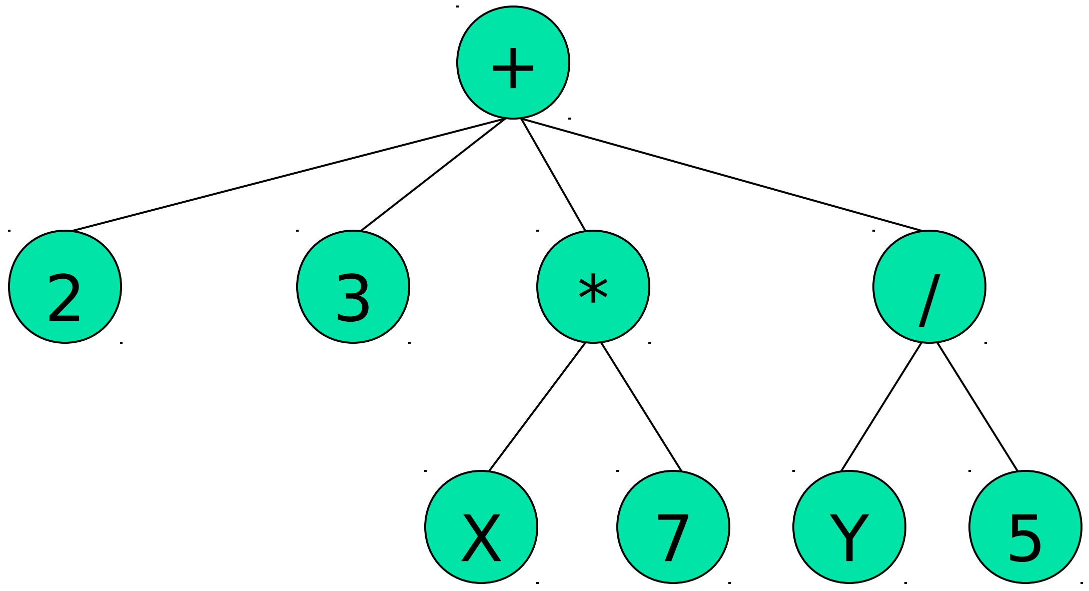
Randomly Generating Programs

(+ 2 3 (* X 7) (/ Y 5))



Mutation

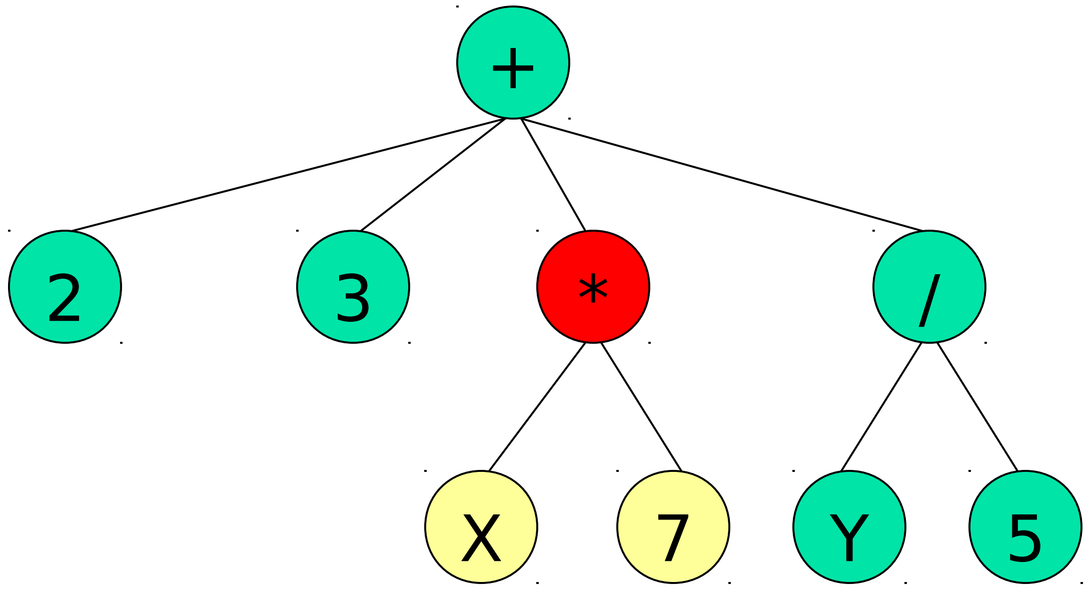
(+ 2 3 (* X 7) (/ Y 5))



Mutation

(+ 2 3 (* X 7) (/ Y 5))

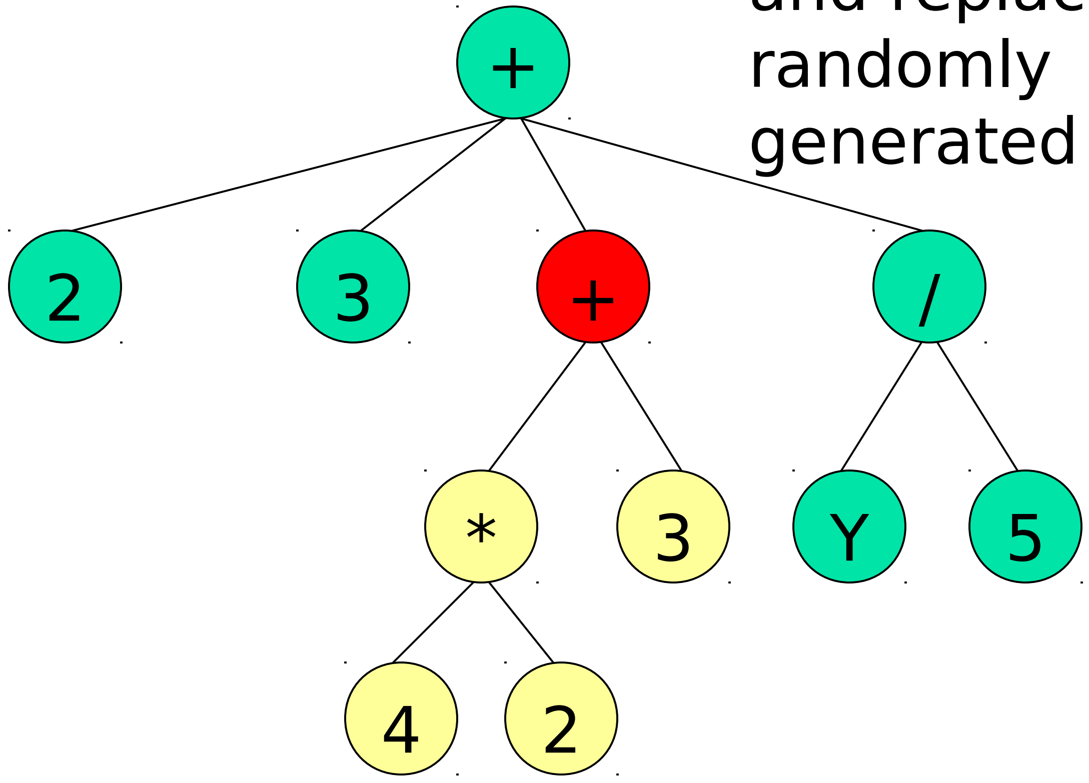
First pick a random node



Mutation

(+ 2 3 (+ (* 4 2) 3) (/ Y 5))

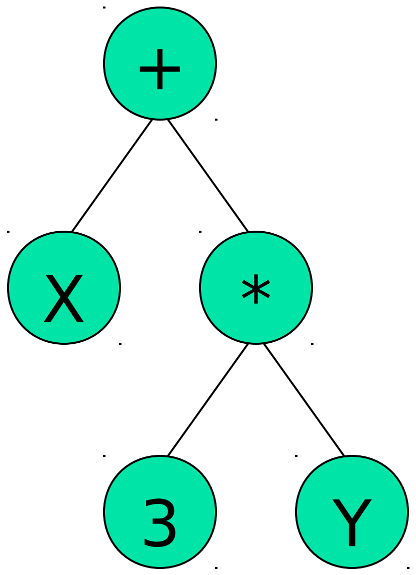
Delete the node and its children, and replace with a randomly generated program

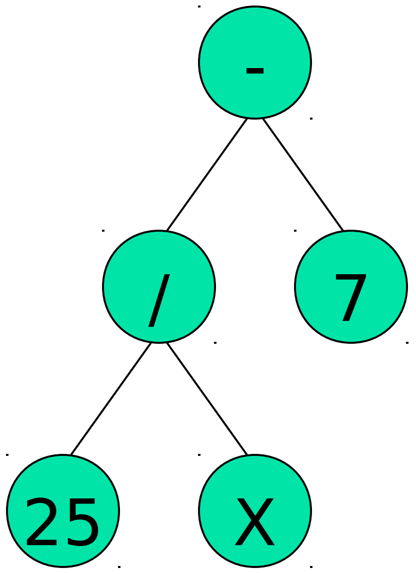


Crossover

(+ X (* 3 Y))

(- (/ 25 X) 7)



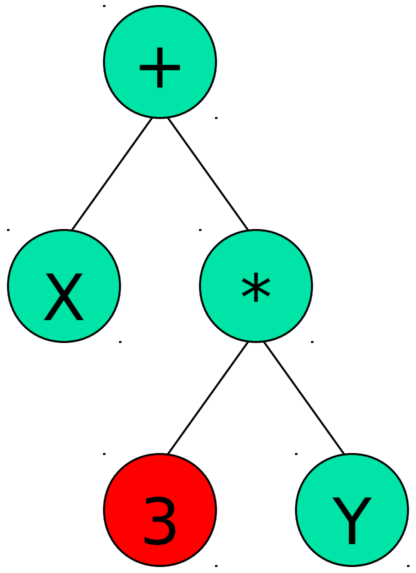


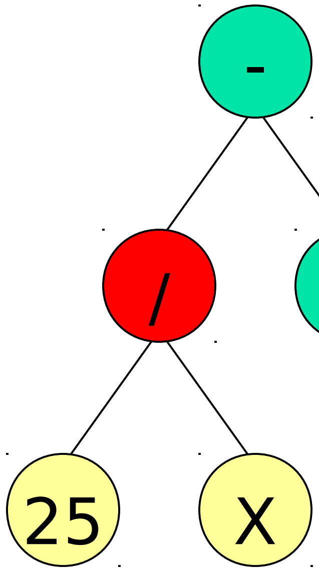
Crossover

(+ X (* 3 Y))

(- (/ 25 X) 7)

Pick a random node in each program



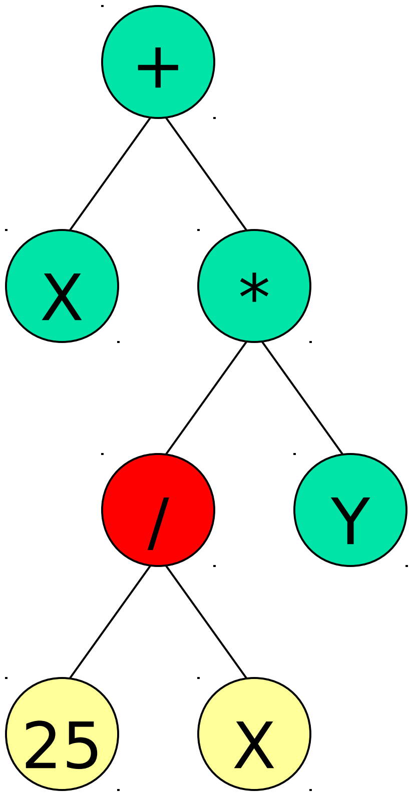


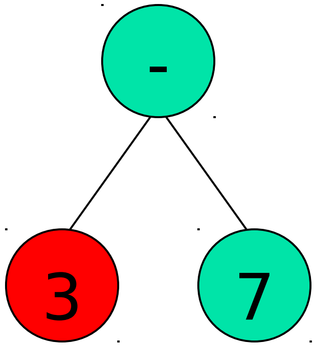
Crossover

(+ X (* (/ 25 X) Y))

(- 3 7)

Swap the two nodes





What About Just Randomly Generating Programs?

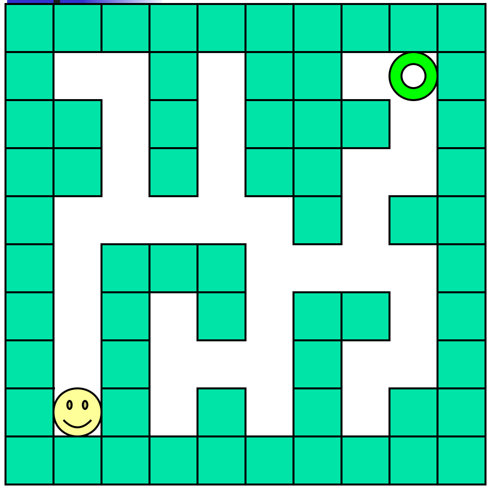
- Is Genetic Programming really better than just randomly creating new functions?
- Yes!
  - Pete Angeline compared the result of evolving a tic-tac-toe algorithm for 200 generations, with a population size of 1000 per generation, against 200,000 randomly generated algorithms
  - The best evolved program was found to be significantly superior to the best randomly generated program [Genetic Programming FAQ, 2002]
- The key lies in using a fitness measure to determine which functions survive to reproduce in each generation

Building a Better Mouse

- Apply Genetic Programming to the problem of navigating a maze
- What are our terminal and function sets?
  - Function Set = {If-Movement-Blocked, While-Not-At-Cheese*}
  - Terminal Set = {Move-Forward, Turn-Left, Turn-Right}

* While-Not-At-Cheese will be used exclusively as the root node of the parse tree

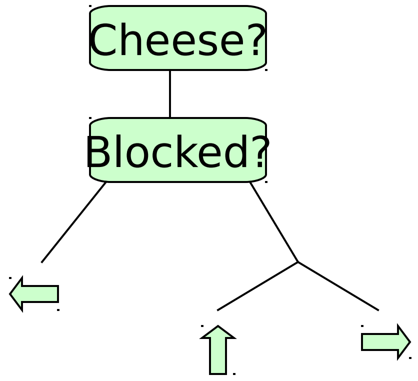
Building a Better Mouse



How to get the starving mouse to the cheese?

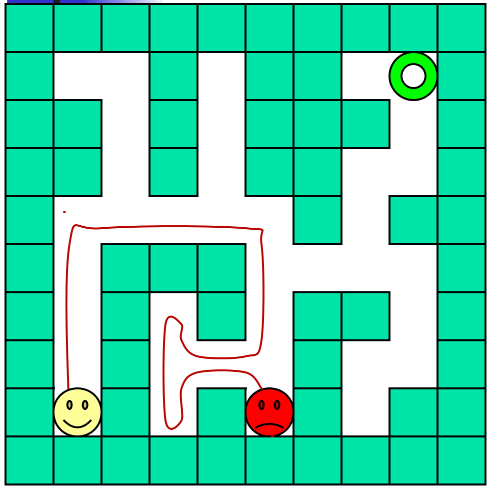
One possible solution:

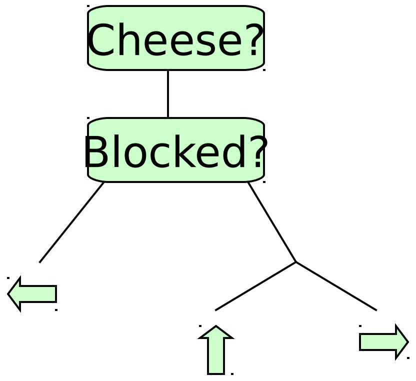
```
While not at the cheese
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Move forward
    Turn right 90 degrees
```



Is there a better solution for this maze? How good is this solution?

Building a Better Mouse



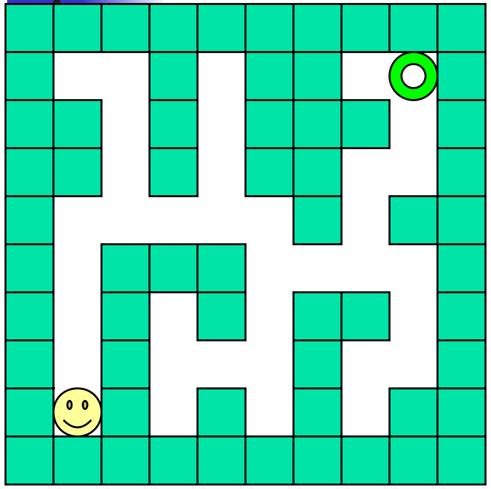


A fitness function:

- Each function and terminal other than the root node shall cost one unit to execute
- If the mouse spends more than 100 units, it dies of hunger
- The fitness measure for a program is determined be executing the program, then squaring the sum of the total units spent and the final distance from the exit
- A lower fitness measure is preferable to a higher fitness measure

Our mouse will die 10 moves from the exit after spending 100 units, so the fitness measure for our program is 12100

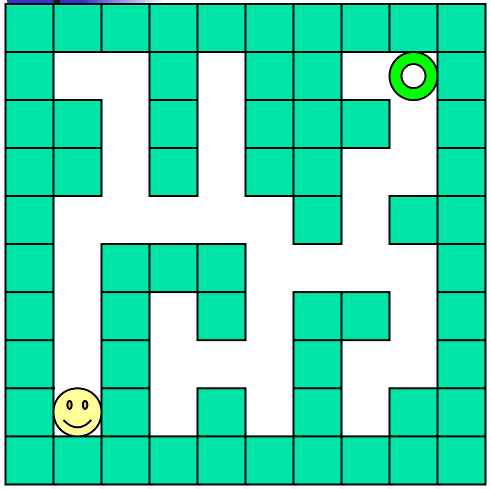
Building a Better Mouse



```
While not at the cheese          (12996)
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Move forward one space

While not at the cheese          (12996)
  Move forward one space
  Turn right 90 degrees
  Turn left 90 degrees
```

Building a Better Mouse



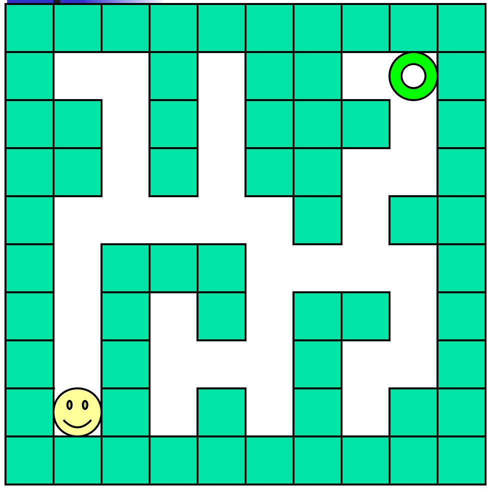
```
While not at the cheese            (12996)
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Move forward one space
```

```
While not at the cheese            (12996)
  Move forward one space
  Turn right 90 degrees
  Turn left 90 degrees
```

Mutation:

```
While not at the cheese            (12996)
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Turn left 90 degrees
```

Building a Better Mouse



```
While not at the cheese          (12996)
   If the way ahead is blocked
      Turn left 90 degrees
   Otherwise
      Move forward one space
```

```
While not at the cheese          (12996)
   Move forward one space
   Turn right 90 degrees
   Turn left 90 degrees
```

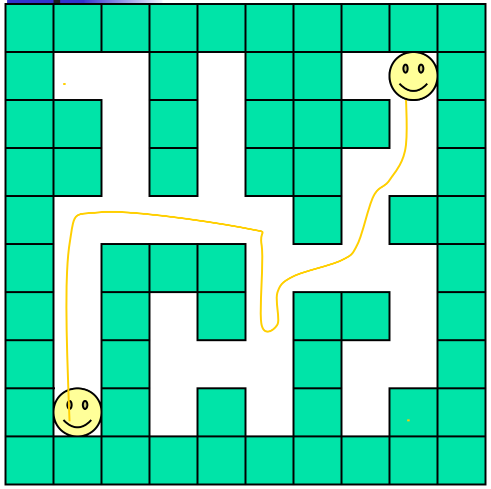
Crossover:

```
While not at the cheese    (11664)
   If the way ahead is blocked
      Move forward one space
      Turn right 90 degrees
   Otherwise
      Move forward one space
```

```
While not at the cheese   (12996)
   Turn left 90 degrees
   Turn left 90 degrees
```

Building a Better Mouse

(after 4202 generations, with 1000 programs per generation)



Is this better?

Fitness measure: 2809

```
While not at the cheese
  If the way ahead is blocked
    Turn right 90 degrees
    Move forward one space
    Move forward one space
    Move forward one space
  Otherwise
    Move forward one space
    Turn right 90 degrees
    Move forward one space
    Move forward one space
    Turn left 90 degrees
    If the way ahead is blocked
      Turn left 90 degrees
    Otherwise
      Move forward one space
```

Real World Applications

- Lockheed Martin Missiles and Space Co. - Near-Minimum-Time Spacecraft Maneuvers [Howley, 96]
- GP applied to the problem of rest-to-rest reorientation maneuvers for satellites
- Optimal time solution is a vector of nonlinear differential equations, which are difficult to solve
- An approximate solution is necessary for a real-time controller
- Results: Rest-to-Rest Maneuver Times (8 test cases)
  - Optimal Solution: 287.93 seconds
  - Expert Solution: 300.3 seconds
  - GP Solution: 292.8 seconds

Real World Applications

- Symbolic Regression
- Problem: Given a set of data points, find a mathematical model

http://alphard.ethz.ch/gerber/approx/default.html

Real World Applications

- Neural Network Optimization [Zhang, Mühlenbein, 1993]
- Image Analysis [Poli, 1996a]
- Generation of a knowledge base for expert systems [Bojarczuk, Lopes, Freitas, 2000]
- Fuzzy Logic Control [Akbarzadeh, Kumbla, Tunstel, Jamshidi, 2000]
- Hardware Evolution (Field-Programmable Gate Array) [Thompson, 1997]

Application of Genetic Programming for Multicategory Pattern Classification

Kishore, J.K., Patnaik, L.M., Mani, V., Agrawal, V.K.,

IEEE Transactions on Evolutionary Computation,

Vol. 4, No. 3, pp. 242-258, 2000

조 동 연

Introduction

Multicategory Pattern Classification

- An input feature vector of *m* dimension is classified as belonging to one of the *n* classes.
- Maximum likelihood classifier (MLC)
  - Mean vectors and covariance matrices
  - Likelihood values are computed for each class for a given input vector.
  - Drawback: distance-based approach, assuming a normal distribution
- Neural network
  - Multilayer: *m* inputs, *n* outputs
  - The given input vector is assigned for the class of the maximum output.
  - Drawback: training time, unknown optimal structure, opaque results

Genetic Programming

- Functions and Terminals
  - F: arithmetic, mathematical, Boolean, conditional operator
  - T: features
- Applying GP to pattern classification
  - No *a priori* knowledge is needed about the statistical distribution of the data or no assumption is made as in MLC.
  - GP can operated directly on the data in their original form.
  - GP can detect the underlying but unknown relationship that exists among data.
  - GP can discover the most important discriminative features of a class.

- The GP paradigm is extended to the *n*-class problem.
  - As a typical GP expression returns a value (+1 or –1) for a two-class problem, how does one apply GP for the *n*-class pattern classification problem?
  - What should be the fitness function of the GP expressions?
  - How does the choice of a functions set affect the performance of GP-based classification?
  - How should training sets be created for evaluating fitness during the evolution of GP expressions?
  - How does one improve learning of the underlying data distributions in a GP framework?
  - How should conflict resolution be handled before assigning a class to the input feature vector?
  - How does GP compare with other classifiers?

GP-Based Classification

Formulation of the *n*-Class Problem as *n* Two-class problems

- For example, a five-class problem
  - n_j: the number of samples that belong to class j (j = 1, …5)
  - N_j: the numb[redacted] not belong to class j

- Geneti[redacted]ssion (GPCE)

- Fitness Measure

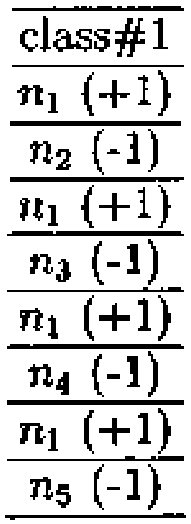
[redacted]

- Choice of Function Set
  - Arithmetic (A): +, -, ×, ÷
  - Arithmetic and logical (AL): +, -, ×, ÷, *IFLTE*
  - Arithmetic and nonlinear (ANL): +, -, ×, ÷, *SINE*
  - Arithmetic, logical, and nonlinear (ALNL): +, -, ×, ÷, *IFLTE*, *SINE*

Creation of Training Sets

- Highly skewed training set
  - The number of samples belonging to one class (whose desired output is +1) is outnumbered by the samples belonging to all other classes (whose desired output is –1).
  - For example, n_1 = 100, N_1 = 400

- Interleaved Data Format
  - Positive examples are repeated
    - The samples belonging to the true class are alternately placed between samples belonging to other classes.
    - Since the number of samples in the training set is increased, the time taken for evaluating fitness also increases. [n_1+N_1 ↔

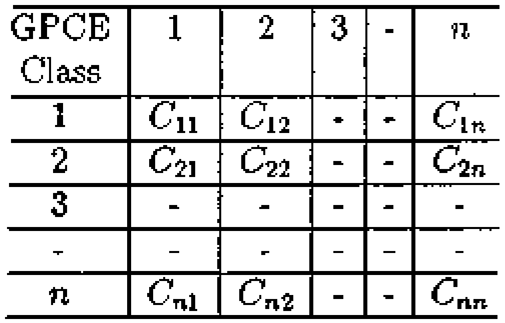


class#1
n_1 (+1)
n_2 (-1)
n_1 (+1)
n_3 (-1)
n_1 (+1)
n_4 (-1)
n_1 (+1)
n_5 (-1)

- Incremental Learning
  - Global learning
    - GP tries to learn the entire training set at every stage of the evolution.
  - Incremental learning global
    - A subset of the training is fed, and the size of the subset is increased gradually over time to cover the entire training data.
    - The basic motivation is to improve learning during evolution as it is easier to learn a smaller task, then to progress from a smaller task to a bigger task.

Conflict Resolution

- For every input in the validation set, all of the GPCEs are applied.
  - *n*-dimensional result vector containing +1 or –1
  - Three situations
    - Only one GPCE returns a value of +1.   [-1, +1, -1, -1, -1]
    - More than one GPCE return a value of +1. [+1, -1, +1, -1, -1]
    - All GPCEs return a value of –1.   [-1, -1, -1, -1, -1]
- Class count matrix
  - All of the GPCEs are applied on all of the samples in the training set.
  - C_{ij}: the number of samples of class *i* for which GPCE$_j$ returns



GPCE Class	1	2	3	-	n
1	C_{11}	C_{12}	-	-	C_{1n}
2	C_{21}	C_{22}	-	-	C_{2n}
3	-	-	-	-	-
-	-	-	-	-	-
n	C_{n1}	C_{n2}	-	-	C_{nn}

- Strength of association (SA) measure for each GPEC
  - It indicates the degree to which a GPCE can recognize samples belonging to its own class, and reject samples belonging to other c [redacted]
  - Since the SA alone may not be enough for classification, heuristic rules are introduced.
- Automatic Discovery of Discriminant Features
  - A particular feature may be important for one class, and not so for another class.
    - GP has the capability to automatically discover the underlying data relationship among these features, and discard the remaining features during evolution.

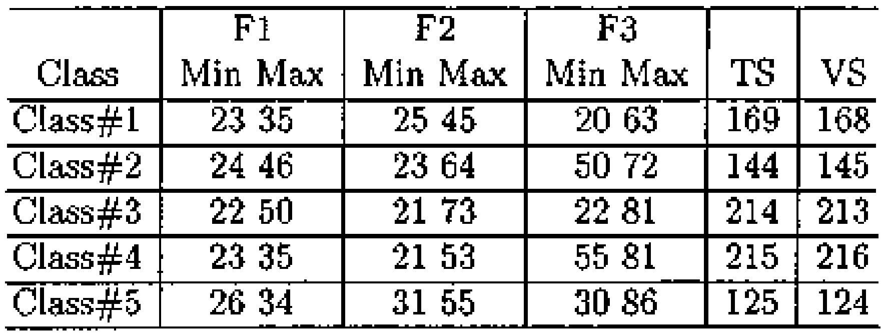
Statistical Measures for Validating a Classifier

- Classification Matrix (CM)
  - GPCEs are applied to the validation set.
    - The dimension of the CM is $n \times (n+1)$.
      - Rejection class
    - A entry q_{ij} in the CM shows how many samples belonging to class i have been classified as class j.
- Class-level Performance
  - Percentage classification (for class i) = q_{ii}/n_i
  - Polarization measure = $q_{ii}/\Sigma\, q_{ji}$
- Global Index
  - Average accuracy = $\Sigma(q_{ii}/n_i)/n$
  - Overall accuracy = $\Sigma q_{ii}/N_v$

Experimental Results for GP-Based Classification

Experimental Setup

- Data1
  - A data set containing five classes taken from three bands of remotely sensed satellite data.
  - Three feature vectors *F*1, *F*2, *F*3

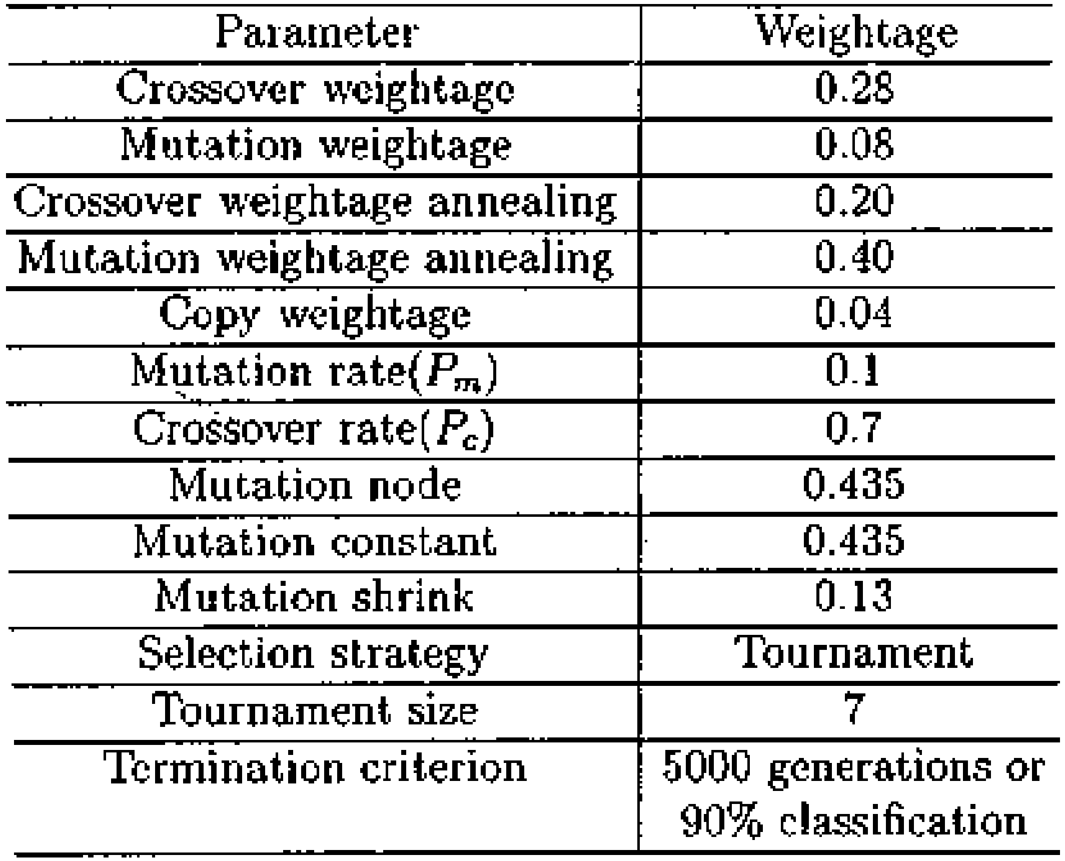


Class	F1 Min Max	F2 Min Max	F3 Min Max	TS	VS
Class#1	23 35	25 45	20 63	169	168
Class#2	24 46	23 64	50 72	144	145
Class#3	22 50	21 73	22 81	214	213
Class#4	23 35	21 53	55 81	215	216
Class#5	26 34	31 55	30 86	125	124

- Data2
  - Iris data set (3 classes, 50 samples for each class)

Tool

- GPQUICK



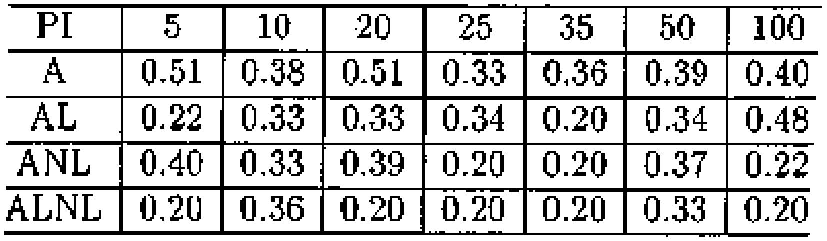
Parameter	Weightage
Crossover weightage	0.28
Mutation weightage	0.08
Crossover weightage annealing	0.20
Mutation weightage annealing	0.40
Copy weightage	0.04
Mutation rate(P_m)	0.1
Crossover rate(P_c)	0.7
Mutation node	0.435
Mutation constant	0.435
Mutation shrink	0.13
Selection strategy	Tournament
Tournament size	7
Termination criterion	5000 generations or 90% classification

Skewness in the Training Set

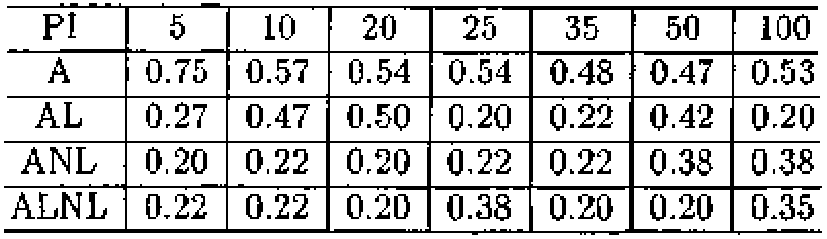
- For class 1
  - 169 (+1), 698 (-1)
  - Interleaved data format: 676 (+1), 698 (-1)
  - Training time: $\times$1.58
- For class 2
  - 144 (+1), 723 (-1)
  - Interleaved data format: 576 (+1), 723 (-1) $\rightarrow$ 720 (+1), 723 (-1)
- The general idea of interleaved data format
  - Properly balancing the number of samples belonging to one class with the number of samples belonging to the other classes

Evolution for Different Function Sets

- Effect of function set
  - A, AL, ANL, ALNL
- Incremental and global learning
  - Percentage increment (PI)
  - Augmenting the training set after every 200 generations



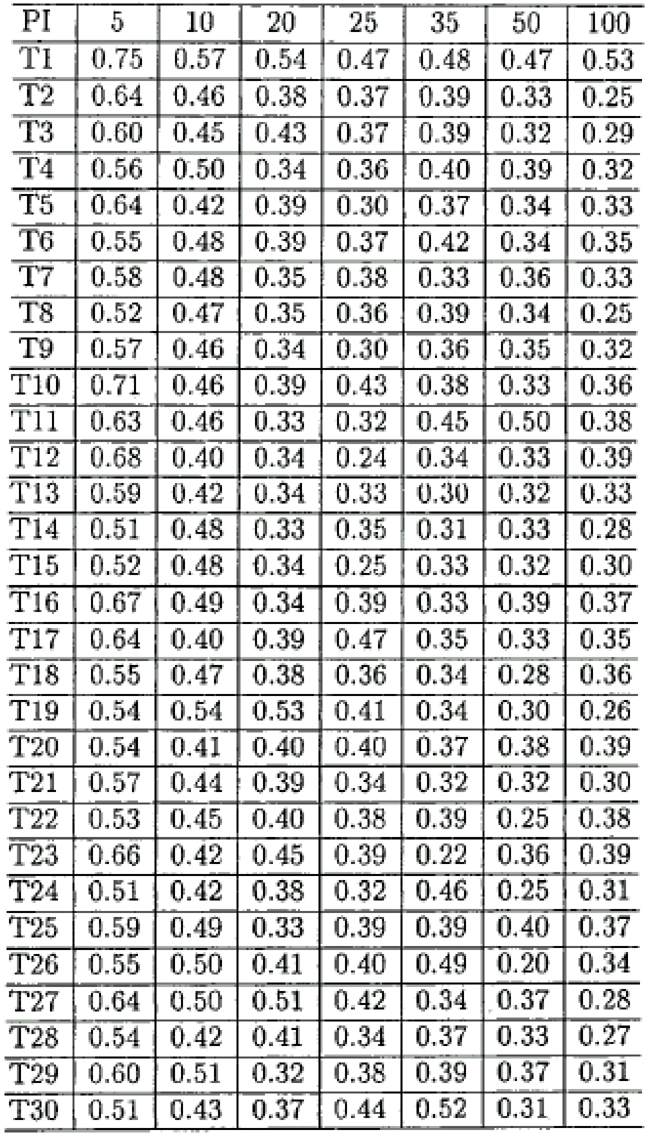
PI	5	10	20	25	35	50	100
A	0.51	0.38	0.51	0.33	0.36	0.39	0.40
AL	0.22	0.33	0.33	0.34	0.20	0.34	0.48
ANL	0.40	0.33	0.39	0.20	0.20	0.37	0.22
ALNL	0.20	0.36	0.20	0.20	0.20	0.33	0.20



PI	5	10	20	25	35	50	100
A	0.75	0.57	0.54	0.54	0.48	0.47	0.53
AL	0.27	0.47	0.50	0.20	0.22	0.42	0.20
ANL	0.20	0.22	0.20	0.22	0.22	0.38	0.38
ALNL	0.22	0.22	0.20	0.38	0.20	0.20	0.35

Average Accuracy for the Skewed (left) and Interleaved (right) Data Set

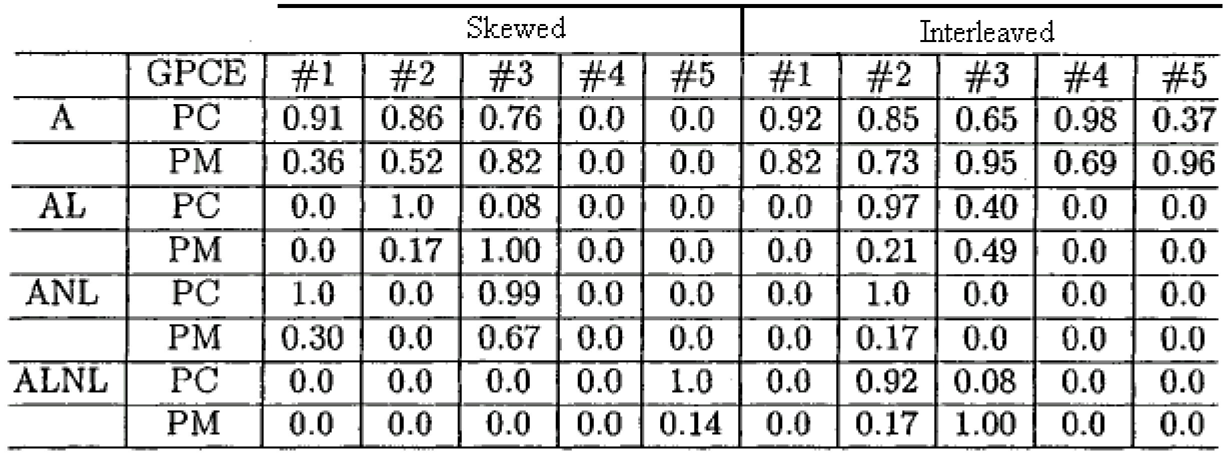
- Some conclusions
  - Incremental learning leads to better classification than global learning.
  - The smaller the percentage increment, the better is the classification.
  - The average accuracy with the arithmetic function set performs better than all other functions sets.
  - There is variation in the classification accuracy in GP-based classification for every trial.



PI	5	10	20	25	35	50	100
T1	0.75	0.57	0.54	0.47	0.48	0.47	0.53
T2	0.64	0.46	0.38	0.37	0.39	0.33	0.25
T3	0.60	0.45	0.43	0.37	0.39	0.32	0.29
T4	0.56	0.50	0.34	0.36	0.40	0.39	0.32
T5	0.64	0.42	0.39	0.30	0.37	0.34	0.33
T6	0.55	0.48	0.39	0.37	0.42	0.34	0.35
T7	0.58	0.48	0.35	0.38	0.33	0.36	0.33
T8	0.52	0.47	0.35	0.36	0.39	0.34	0.25
T9	0.57	0.46	0.34	0.30	0.36	0.35	0.32
T10	0.71	0.46	0.39	0.43	0.38	0.33	0.36
T11	0.63	0.46	0.33	0.32	0.45	0.50	0.38
T12	0.68	0.40	0.34	0.24	0.34	0.33	0.39
T13	0.59	0.42	0.34	0.33	0.30	0.32	0.33
T14	0.51	0.48	0.33	0.35	0.31	0.33	0.28
T15	0.52	0.48	0.34	0.25	0.33	0.32	0.30
T16	0.67	0.49	0.34	0.39	0.33	0.39	0.37
T17	0.64	0.40	0.39	0.47	0.35	0.33	0.35
T18	0.55	0.47	0.38	0.36	0.34	0.28	0.36
T19	0.54	0.54	0.53	0.41	0.34	0.30	0.26
T20	0.54	0.41	0.40	0.40	0.37	0.38	0.39
T21	0.57	0.44	0.39	0.34	0.32	0.32	0.30
T22	0.53	0.45	0.40	0.38	0.39	0.25	0.38
T23	0.66	0.42	0.45	0.39	0.22	0.36	0.39
T24	0.51	0.42	0.38	0.32	0.46	0.25	0.31
T25	0.59	0.49	0.33	0.39	0.39	0.40	0.37
T26	0.55	0.50	0.41	0.40	0.49	0.20	0.34
T27	0.64	0.50	0.51	0.42	0.34	0.37	0.28
T28	0.54	0.42	0.41	0.34	0.37	0.33	0.27
T29	0.60	0.51	0.32	0.38	0.39	0.37	0.31
T30	0.51	0.43	0.37	0.44	0.52	0.31	0.33

Average Accuracy for 30 trials (*T*) with arithmetic function set for different values of PI

- The arithmetic function set with an interleaved data format and incremental learning has given the best classwise performance.
  - When a logical element is used, one of the subexpressions in the LISP *S* expression can return the same value for different inputs.
  - If the SINE function appears in the subexpression followed by a MUL or DIV operator, it is possible for
    val the
    inp



		Skewed					Interleaved				
	GPCE	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5
A	PC	0.91	0.86	0.76	0.0	0.0	0.92	0.85	0.65	0.98	0.37
	PM	0.36	0.52	0.82	0.0	0.0	0.82	0.73	0.95	0.69	0.96
AL	PC	0.0	1.0	0.08	0.0	0.0	0.0	0.97	0.40	0.0	0.0
	PM	0.0	0.17	1.00	0.0	0.0	0.0	0.21	0.49	0.0	0.0
ANL	PC	1.0	0.0	0.99	0.0	0.0	0.0	1.0	0.0	0.0	0.0
	PM	0.30	0.0	0.67	0.0	0.0	0.0	0.17	0.0	0.0	0.0
ALNL	PC	0.0	0.0	0.0	0.0	1.0	0.0	0.92	0.08	0.0	0.0
	PM	0.0	0.0	0.0	0.0	0.14	0.0	0.17	1.00	0.0	0.0

- Analysis of GPCEs
  - GPCE 1
    - (DIV(MUL(DIV(DIV F3-121)(ADD F2-34)(MUL(ADD F2 (DIV –37 (ADD (DIV F2 F2) (SUB –35 F1))))(ADD 71 F3))) (SUB(ADD 109 F2)(SUB 87 (SUB –51 14))))

[redacted]

    - It returns +1 only when *F*2 is greater than 34 and less than 43.
    - *F*2 is the most discriminating feature for class 1.

- GPCE 2

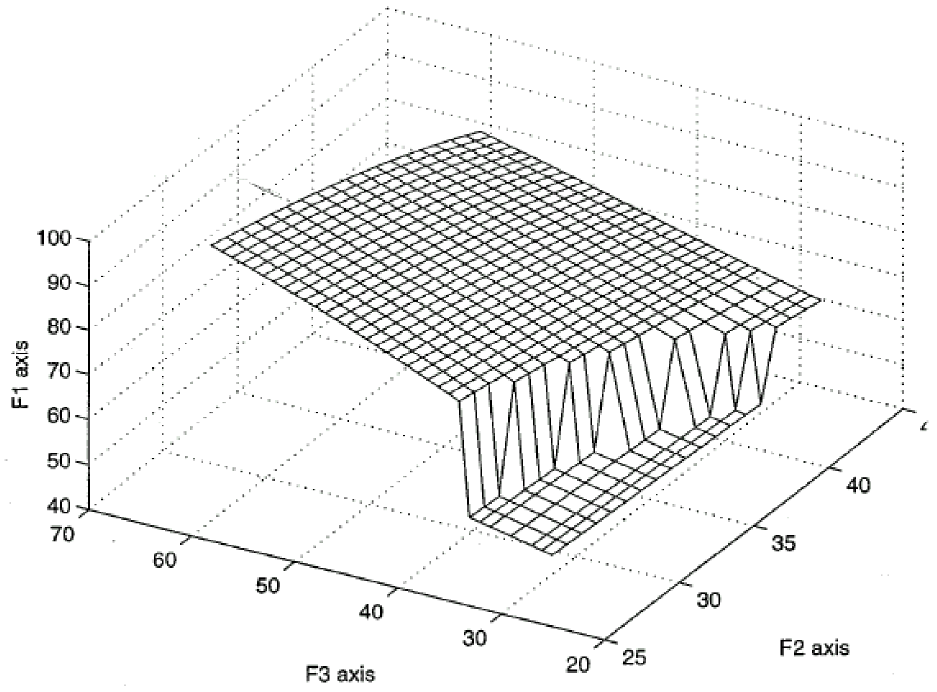
[redacted]

  - It returns +1 when $F2 > 1.031F1$ and $F2 < 36$.
  - $F1$ and $F2$ are discriminant features of class 2.
- GPCE 3

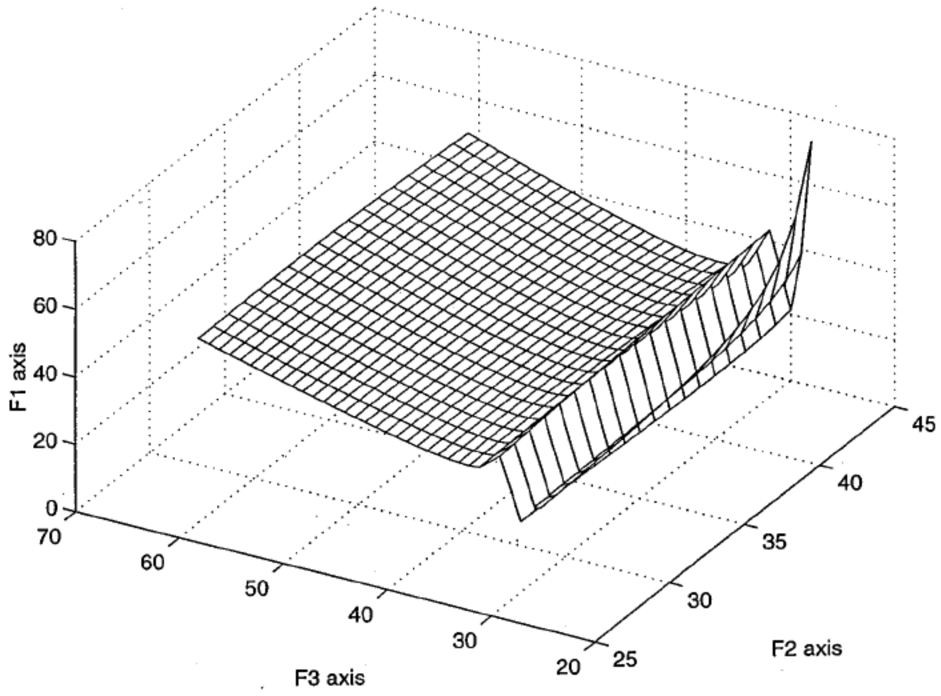
[redacted]

  - It returns +1 only when $F3$ is greater than 61.
  - $F3$ is the discriminant feature for class 3.

- Pictorial Representation of GPCEs
  - [redacted]

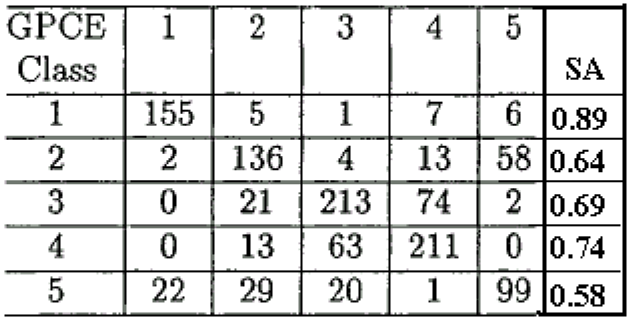


[redacted]

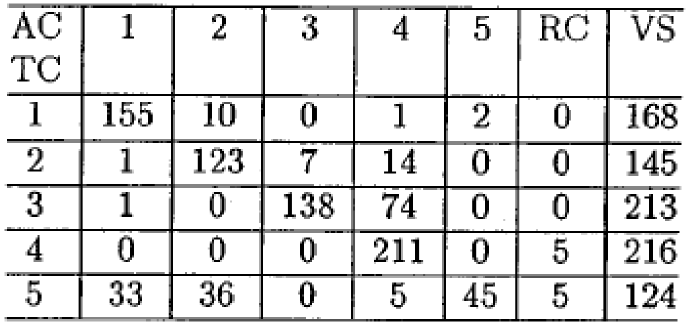


SA Computation and Heuristic Rules

- SA Computation
  - Class count matrix, SAs, and classification matrix based on SA



GPCE Class	1	2	3	4	5	SA
1	155	5	1	7	6	0.89
2	2	136	4	13	58	0.64
3	0	21	213	74	2	0.69
4	0	13	63	211	0	0.74
5	22	29	20	1	99	0.58

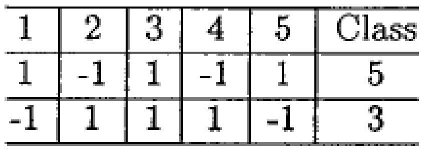


AC TC	1	2	3	4	5	RC	VS
1	155	10	0	1	2	0	168
2	1	123	7	14	0	0	145
3	1	0	138	74	0	0	213
4	0	0	0	211	0	5	216
5	33	36	0	5	45	5	124

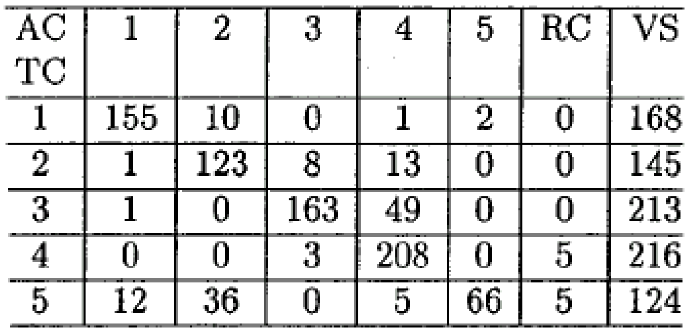
AA=0.752, OA = 0.776

- Conflict resolution based on SA measures
  - For the sample in which 34 < *F*2 < 36 and *F*2 > 1.036F1, both GPCE1 and GPCE2 return a value of +1.
  - The class of the GPCE with the higher SA is assigned to the input, i.e., class 1 is assigned.

- Heuristics
  - If (34 < *F*2 <36 and *F*2 > 1.031*F*1) then, the true class is class 2.
    - This data relationship is expressed by the following result vector [+1 +1 –1 –1 –1].
  - If (34 < *F*2 < 43 and *F*3 > 62), then the true clas
    - [1 –1 1
  - Let α be the user threshold for extraction of a



1	2	3	4	5	Class
1	-1	1	-1	1	5
-1	1	1	1	-1	3



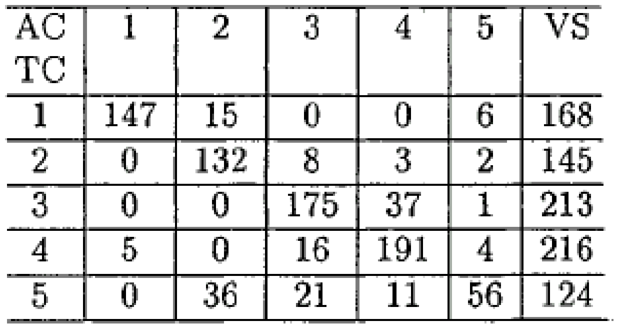
AC TC	1	2	3	4	5	RC	VS
1	155	10	0	1	2	0	168
2	1	123	8	13	0	0	145
3	1	0	163	49	0	0	213
4	0	0	3	208	0	5	216
5	12	36	0	5	66	5	124

Average accuracy = 0.806

Overall accuracy = 0.826

- Performance of the MLC
  - The discriminant function in MLC

[redacted]

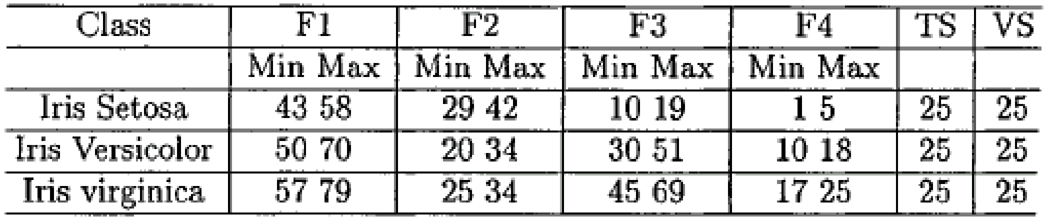


AC TC	1	2	3	4	5	VS
1	147	15	0	0	6	168
2	0	132	8	3	2	145
3	0	0	175	37	1	213
4	5	0	16	191	4	216
5	0	36	21	11	56	124

Average accuracy = 0.789

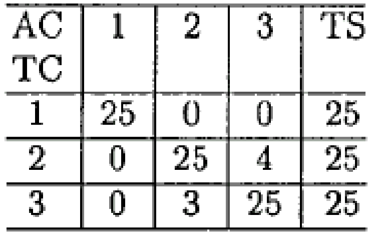
Overall accuracy = 0.809

- GP-Based Classification for Fisher’s Iris Data Set
  - Data
    - Four features, three classes



Class	F1	F2	F3	F4	TS	VS
	Min Max	Min Max	Min Max	Min Max		
Iris Setosa	43 58	29 42	10 19	1 5	25	25
Iris Versicolor	50 70	20 34	30 51	10 18	25	25
Iris virginica	57 79	25 34	45 69	17 25	25	25

set and a validation set

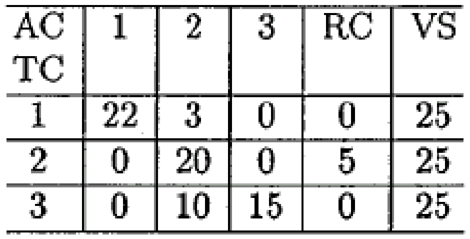


AC TC	1	2	3	TS
1	25	0	0	25
2	0	25	4	25
3	0	3	25	25

Class count

Results

- Skewed

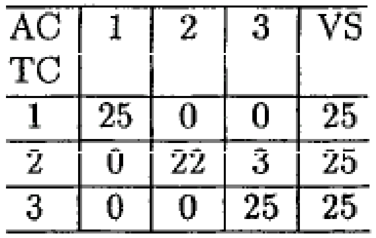


AC TC	1	2	3	RC	VS
1	22	3	0	0	25
2	0	20	0	5	25
3	0	10	15	0	25

Average accuracy = 0.76

Overall accuracy = 0.76

- Interleaved

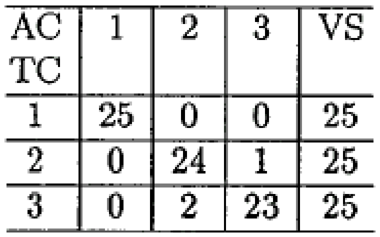


AC TC	1	2	3	VS
1	25	0	0	25
2	0	22	3	25
3	0	0	25	25

Average accuracy = 0.96

Overall accuracy = 0.96

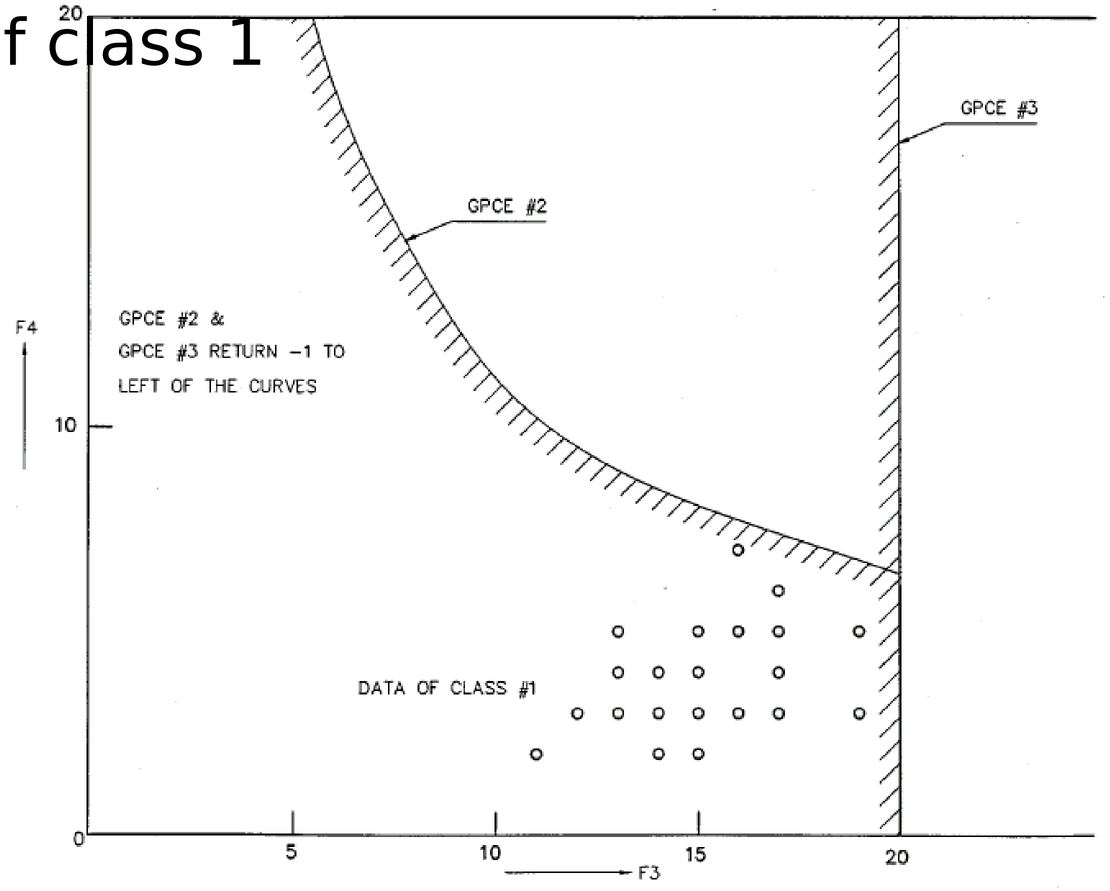
- MLC



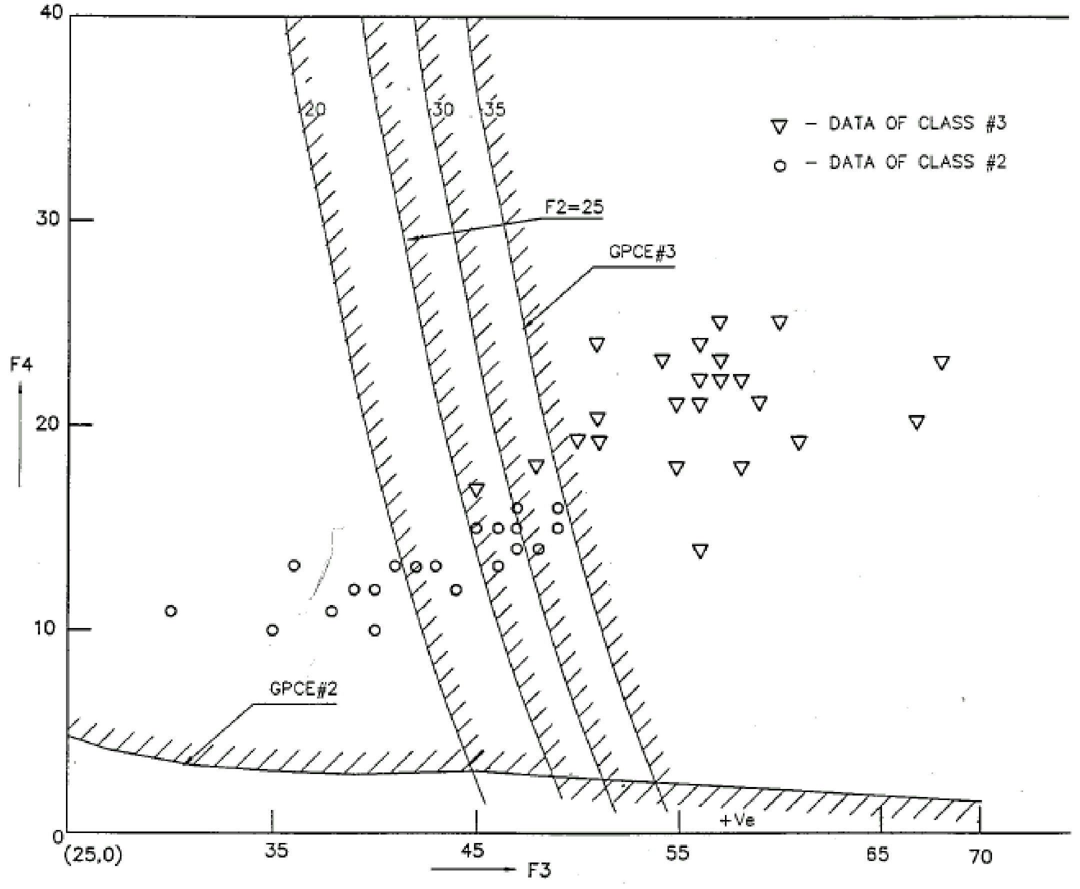
AC TC	1	2	3	VS
1	25	0	0	25
2	0	24	1	25
3	0	2	23	25

Average accuracy = 0.96

- Analysis
  - Behavior of GPCE 2 and GPCE 3 for data of class 1



- Performance of GPCE 3 for different value



Some Important Issues in GP-Based Pattern Classification

- Interleaved Data Format
- Incremental Learning
- Conflict Resolution
- Scope for Heuristic Rules
- Reject Class
- GP and AI-Based Machine Learning

Conclusions

- Applicability of GP to an *n*-class pattern classification problem
  - *n* two-class problem
  - Interleaved data format
  - Incremental learning
  - Arithmetic functions
  - Conflict resolution: SA measure and heuristic rules
- Future works
  - Adaptation variance of GP parameters
  - Discovering any empirical relationship among the data distribution

What’s Next?

- Parallel Distributed Genetic Programming [Poli, 1996b]
    - Operates on graphs rather than parse trees
- Finite State Automata
- Asymetric Recurrent Neural Networks [Pujol, Poli, 1997]

References

- *Genetic Programming FAQ,* 2002. http://www.cs.ucl.ac.uk/research/genprog/gp2faq/gp2faq.html
- Akbarzadeh, M.R., Kumbla, K., Tunstel, E., Jarnshidi, M., “Soft Comuting for Autonomous Robotic Systems”, *Computers and Electrivcal Engineering*, 2002: 26, pp. 5-32.
- Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., “Genetic Programming for Knowledge Discovery In Chest-Pain Diagnosis”, *IEEE Engineering in Medicine and Biology Magazine,* 2000: 19, v. 4, pp. 38-44.
- Howley, B., “Genetic Programming of Near-Minimum-Time Spacecraft Attitude Maneuvers”, *Proceedings of Genetic Programming 1996,* Koza, J.R. et al. (Eds), MIT Press, 1996, pp. 98-109.
- Koza, J., “Genetic Programming as a Means for Programming Computers by Natural Selection“, 1994.
- Poli, R., “Genetic Programming for Image Analysis”, *Proceedings of Genetic Programming 1996,* Koza, J.R. et al. (Eds), MIT Press, 1996, pp. 363-368.
- Poli, R., “Parallel Distributed Genetic Programming”, *Technical* report, The University of Birmingham, School of Computer Science, 1996.
- Pujol, J.C.F., Poli, R., “Efficient Evolution of Asymetric Recurrent Neural Networks Using a Two-dimensional Representation”, 1997.
- Thompson, A., “Artificial Evolution in the Physical World”, *Evolutionary Robotics: From Intelligent Robots to Artificial Life*, Gomi, T. (Ed.), AAI Books, 1997, pp. 101-125.
- Zhang, B.T., Mühlenbein, H., “Genetic Programming of Minimal Neural Nets Using Occam’s Razor”, *Proc. Of 5th Int. Conf. On Genetic Algorithms,* 1993, pp. 342-349.