# CHAPTER II
# Genetic Programming

# Genetic programming

- More recent evolutionary algorithms (work of **John Koza** in the 90s).
- PG searches the program space for the one that is highly suited to solving a given problem.
- Any computer program is a sequence of operations (functions) applied to values (arguments); different programming languages differ in the types of instructions and operations, in addition to different syntactic constructions.
- PG manipulates programs using genetic operators, they are treated as data to be transformed which, once modified, become new programs. A language well suited to this kind of manipulation is **LISP** .

# Genetic Programming (GP)

- Definitions:
  - **automatic program generation ( *J. Koza* )**
  - **automatic generation of behaviors represented by executable programs ( *P. Angeline* ).**
- These are "strong" definitions because they do not specify "by an evolutionary approach", so why "genetic"?
- We can rather speak of “automatic programming” (J. Koza, W. Banzhaf, ...).
- However, a majority of works are inspired by the philosophy of genetic algorithms .

# Some historical milestones

- 1958, Friedberg: tests of random "mutation" of instructions in a program, attribution of "credits" to the most efficient program instructions.
- 1963, Samuel: use of the term “machine learning” in the sense of automatic programming.
- 1966, Fogel, Owen, Walsh: use of finite state automata for behavior prediction tasks; new individuals obtained by selection of efficient "parents" to which mutations are applied. No crossover.

# Some historical milestones (continued)

- 1985, Cramer: Use of expressions represented in tree form. Cross-over between subtrees.
- 1986, Hicklin: Evolution of game programs in LISP. Efficient selection of "parents", combinations of common subtrees or those present in one of the parents and random subtrees.
- 1989, 1992, Koza: Systematization and demonstration of the interest of this approach for many problems. Definition of a standard paradigm in the book "Genetic Programming. On the Programming of Computers by Means of Natural Selection" [Koza, 1992].

# The “standard” paradigm

- The PG “à la Koza”:
  - programs structured in tree expressions.
  - definition of a set of primitive functions and terminals, the only authorized constituents of expressions.
  - single return type for all expressions.
  - evolution via the mechanisms of subtree cross-over, weighting to minimize leaf exchange in favor of larger subtree exchange.
  - very reduced, if not non-existent, role of random mutations.
  - tree depth limitation due to implementation constraints
- In this model, PG can be described as an instantiation of the genetic algorithm paradigm

# PG as an instantiation of AGs

AG: population of solutions

PG: program population

- We find the “ingredients” of the AGs:
  - representation problem
  - quality measurement (fitness)
  - selection pressure
  - use of a population
  - exchange of information between individuals
  - ...

# Representation of solutions

- Example program:

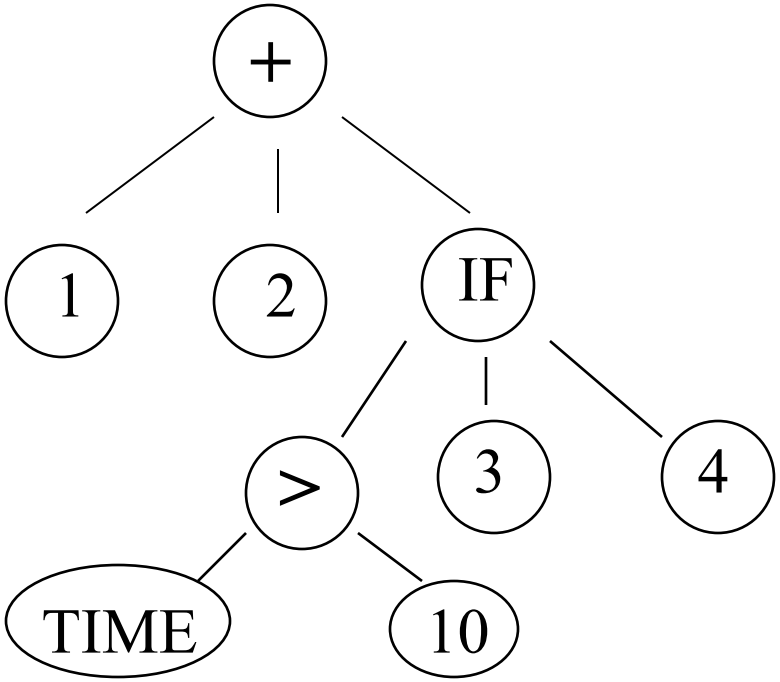
```
 int toto (void)
{
int tmp1, tmp2;
tmp1 = 1 + 2;
if ( TIME > 10 )
tmp2 = 3;
else
tmp2 = 4;
return tmp1 + tmp2;
}
```

# LISP tree representation

- "LISP" expression equivalent to the program:

(+ 1 2 (IF (TIME > 10) 3 4))

- TREE :




- An equivalent implementation is of course possible in more efficient languages (C, C++, Java, ...)

# Terminology (1)

- The terminals (leaves of the tree):
  - pseudo-variables containing the program inputs
  - constants, fixed based on preliminary knowledge of the problem, or randomly generated ( *random ephemeral constants* )
  - functions without arguments but with side effects
  - ordinary variables ( *Note: the functional approach to LISP often dispenses with the presence of variables)*

# Terminology (2)

- Functions or operators (internal nodes of the tree):
  - example: boolean, arithmetic, transcendental, side-effect functions (assignment of variables, movement of a robot, etc.), functions implementing control structures: alternative, loop, call of routines, etc.
  - prefer a set of functions that is small and well adjusted to the problem domain, to reduce the search space. Be careful not to reduce it too much, otherwise you will lose the possibility of finding interesting solutions!

# LISP 101

- Symbol-oriented data structure language. The basic structures are **atoms** and **lists** .
- An atom is the smallest indivisible element in LISP syntax ( eg . the number 21, the symbol X, or the string “This is a string”).
- A list is an object composed of atoms and/or other lists.
- LISP lists are written as an ordered collection of items enclosed in a pair of parentheses.

  So the list

( – (* AB) C)

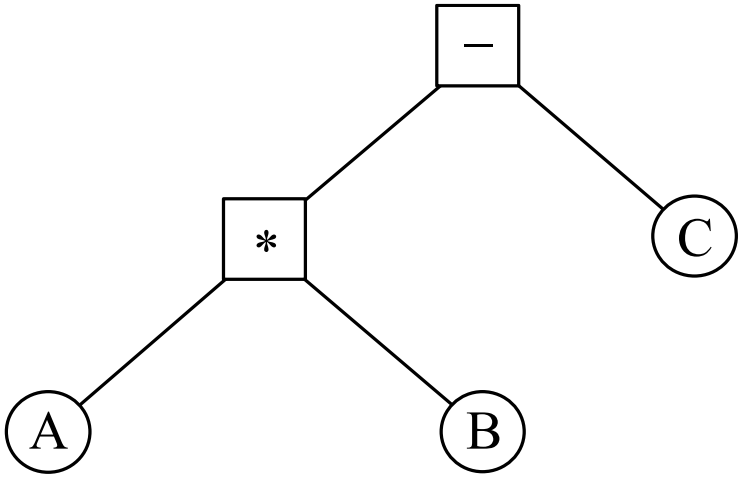
request to subtract two arguments, the list (*AB) and the atom C. But first , he you have to multiply atoms A and B.

# LISP 101

- Atoms and lists are called symbolic expressions or " **S-expressions"** .
- All data and programs are S-expressions, which allows the language to treat programs as data.
- In particular, LISP programs can self-modify or generate new programs, making the language attractive for genetic programming.

- Any S-expression can be represented by a tree.

S-expression ( – (*AB) C)



# How to apply genetic programming

- Before applying genetic programming to a problem, *five preliminary steps must be completed* :

1. Define the terminals
   Program Entries
2. Choose which functions to use
   Arithmetic operations, program expressions, subroutines, mathematical functions, etc.
3. Define the adaptation function
   Often an error function
4. Choose execution parameters
   Same as for GA
5. Choose the method for selecting the best result
   Usually the best program at any given time

# Initial population

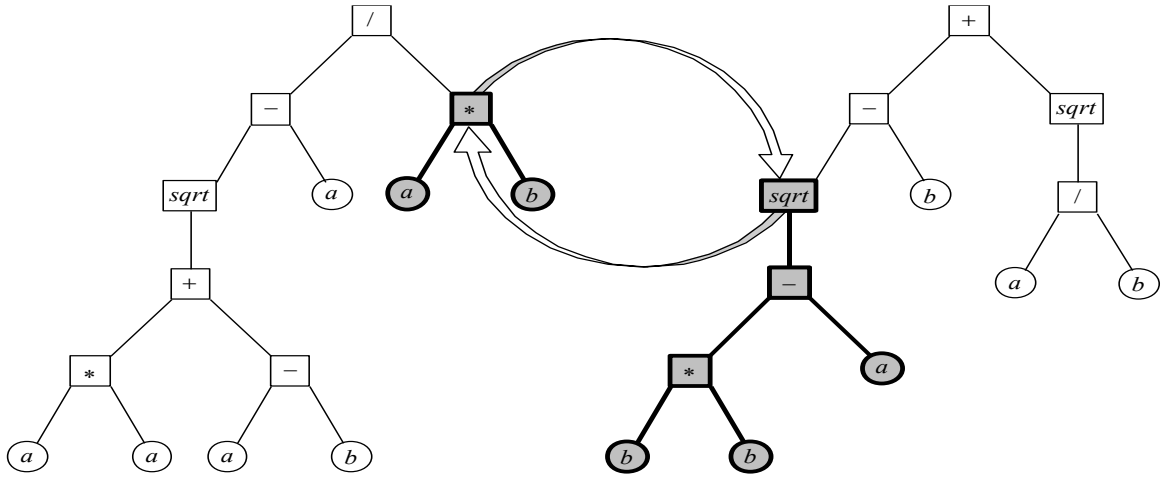
- A maximum depth is set for trees.
- Creation of random trees by 2 main methods:
  - “grow”: each node is pulled into the set {terminals} + {functions}
    - ⇨ the trees are irregularly shaped
  - "full": you can only pull a terminal when you are at maximum depth
    - ⇨ balanced and “full” trees
- A summary, the “ramped half & half” method:
  - we will fairly generate trees of regularly staggered depths:

    2, 3, 4, …, maximum
  - at each depth, one half is generated by the "full" method, the other by the "grow" method
    - ⇨ The goal is to get more variability in the population. This is the preferred method at present.

# Adaptation and selection

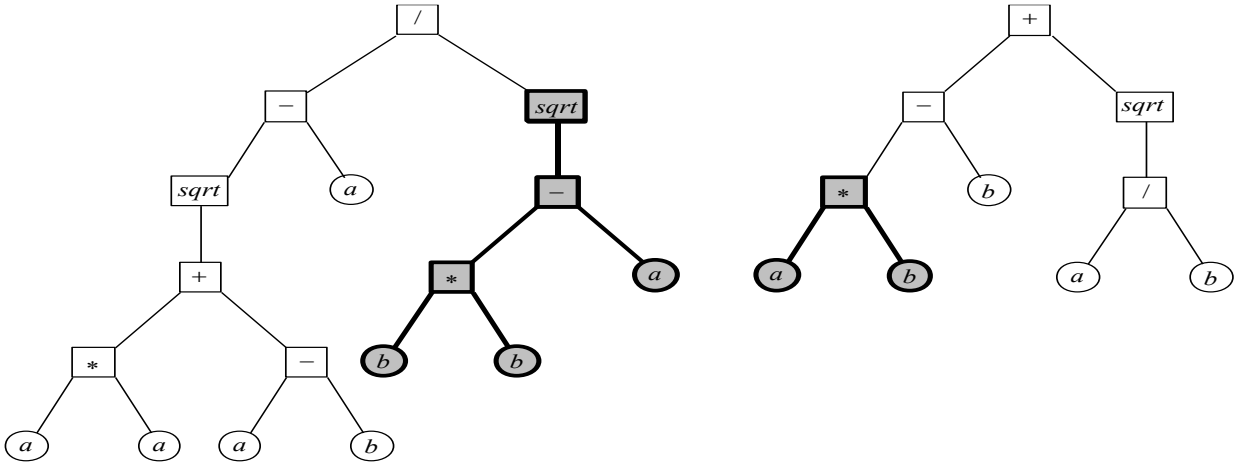
- The choice of the adaptation function is done in a similar way to the AGs
- We also find the selection methods used in the AGs:
  - selection proportional to the degree of adaptation, with possible normalization (“scaling”)
  - selection based on the rank of the individual in the population (“ranking”)
  - tournament selection: the most common, because it is fast and easily parallelizable

# Genetic Operators: Crossing




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

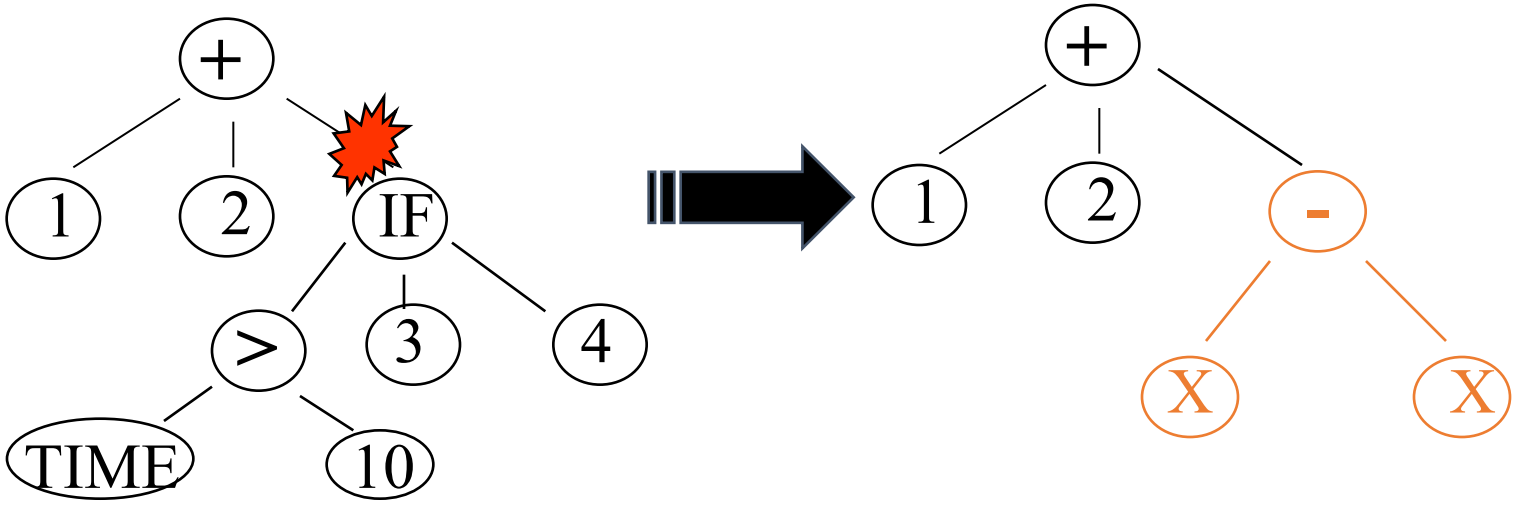
(+ (− (*sqrt* (− (* *b b*) *a*)) *b*) (*sqrt* (/ *a b*)))




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

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

# Genetic Operators: Mutation




- Destruction of a subtree
- Replacement with a random subtree, created as when generating the initial population.

# Note on genetic operators

- Crossover or mutation can transform any argument subtree of a function.
  - ⇨ Functions should be able to accept all kinds of values as arguments, and it is best if they all have the same return value type ( *closure property* )

  Example: Replace standard division with "protected" division which returns 0 or a large integer when divided by 0.

# An example of implementation

- We want to find a program that calculates the function

- C =…….
- We have the following 10 adaptation cases, chosen at random from the domains of variables a and b:

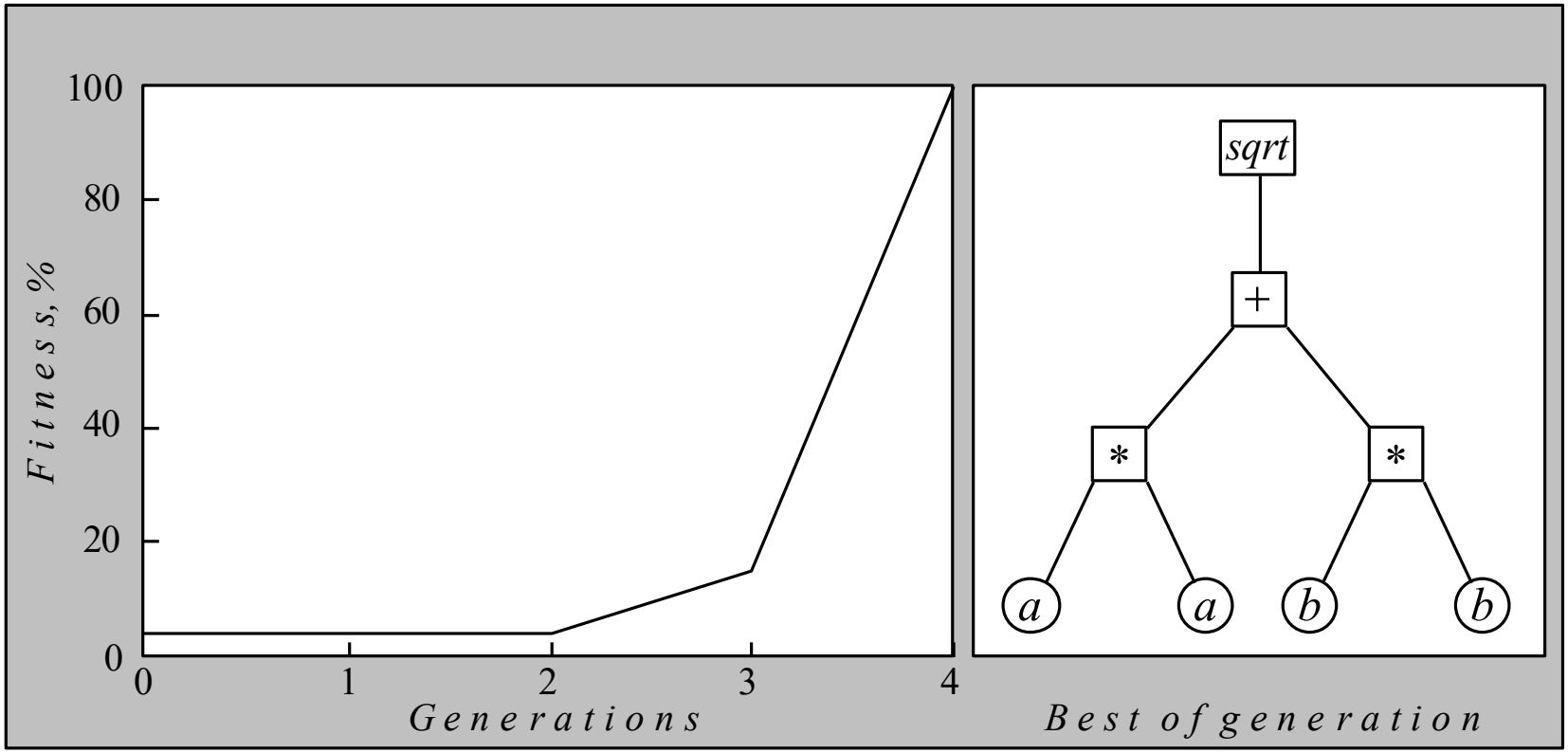
| *a* | *b* | *c* | *a* | *b* | *c* |
|---|---|---|---|---|---|
| 3 | 5 | 5.830952 | 12 | 10 | 15.620499 |
| 8 | 14 | 16.124515 | 21 | 6 | 21.840330 |
| 18 | 2 | 18.110770 | 7 | 4 | 8.062258 |
| 32 | 11 | 33.837849 | 16 | 24 | 28.844410 |
| 4 | 3 | 5.000000 | 2 | 9 | 9.219545 |

# The 5 preliminary steps

1. Terminal identification: a and b
2. Choosing which functions to use: +, -, *, / and sqrt
3. Definition of the adaptability function: sum of squared errors on all cases between the calculated result and that given by the previous table
4. Parameters: Population size and number of generations
5. Method of selecting the best program: the best in each generation.

Once these steps are completed, an initial population of candidate programs is randomly generated and the reproduction cycle begins by applying the operations of crossing, mutation and cloning to each generation.

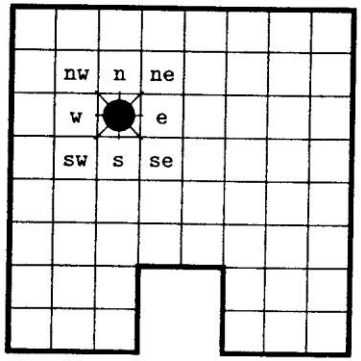
# History of the best s-expression




- The algorithm converges to the correct solution in 5 iterations

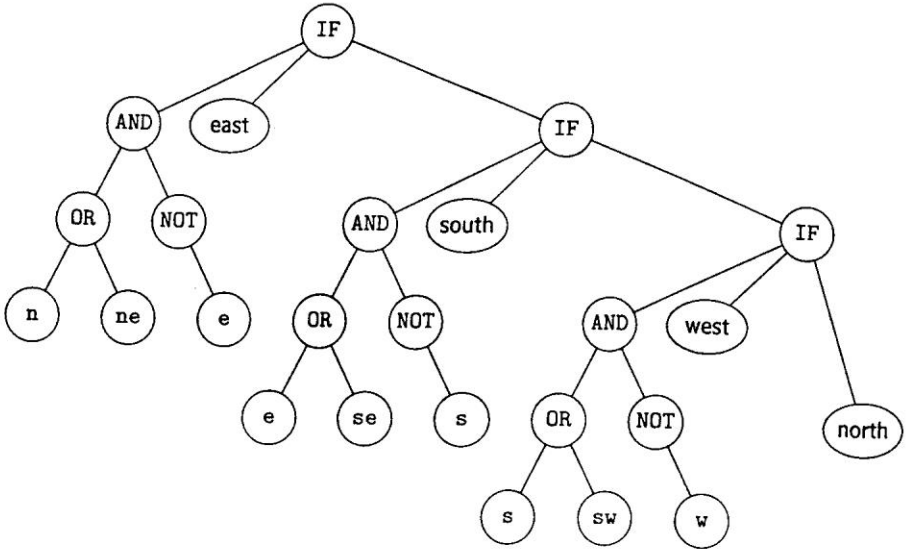
# A more advanced example

- the problem :




A robot must follow the indicated outline; find the sequence of instructions to follow.

- a solution program



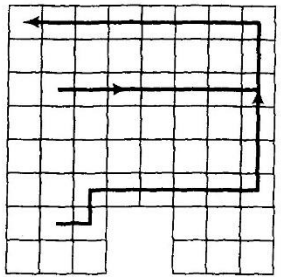

```
(IF (AND (OR (n) (ne)) (NOT (e)))
    (east)
    (IF (AND (OR (e) (se)) (NOT (s)))
        (south)
        (IF (AND (OR (s) (sw)) (NOT (w)))
            (west)
            (north))))
```

- Authorized functions:

**`if/3`**
**`and/2 or/2 not/1`**
**`north south east west`** *(travel)*
**`s se sw e ne nw nw`** *(obstacle tests ≈ sensors)*

Ref.: Artificial intelligence: A new Synthesis by Nils J. Nilsson

- population = 5000 individuals (initially random)
- selection by
  - tournament (10%)
  - crossing (90%)
- Adaptation: we test the program on 10 trials
  value = number of cases of the contour explored (max = 320)

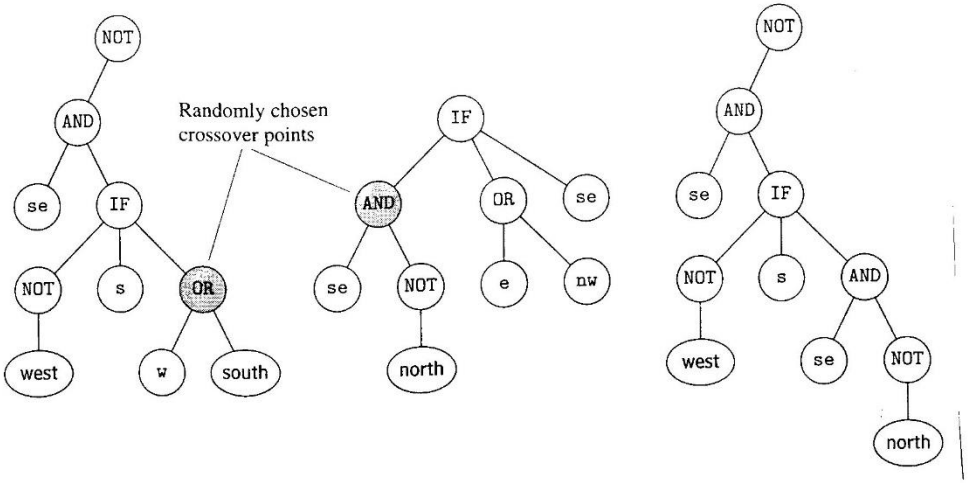


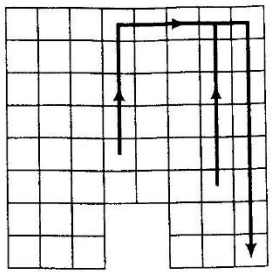
```
(AND (NOT (NOT (IF (IF (NOT (nw))
                       (IF (e)(north) (east))
                       (IF (west)(0) (south))
                   (OR (IF (nw)(ne)(w))
                       (NOT (sw)))
               (NOT (NOT (north))))))
     (IF (OR (NOT (AND (IF (sw)(north)(ne))
             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

**Figure 4.5**

The Most Fit Individual in Generation 0

crossing operation

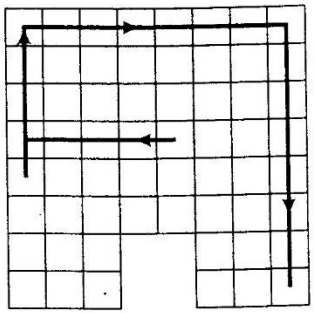




```
(NOT (AND (IF (ne)
              (IF (se)(south)(east))
              (north))
          (NOT (NOT (e)))))
```

**Figure 4.6**

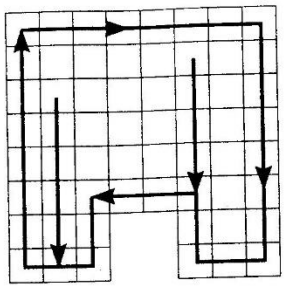
The Most Fit Individual in Generation 2



```
(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))))
```

**Figure 4.7**

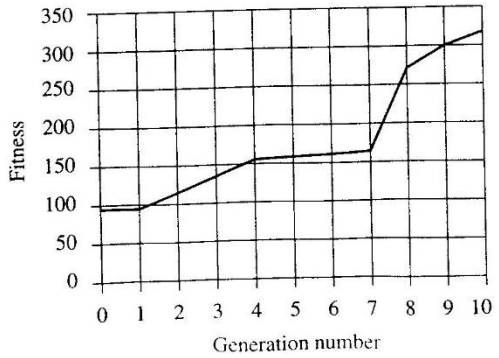
The Most Fit Individual in Generation 6



```
(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)))))))
```

**Figure 4.8**

The Most Fit Individual in Generation 10



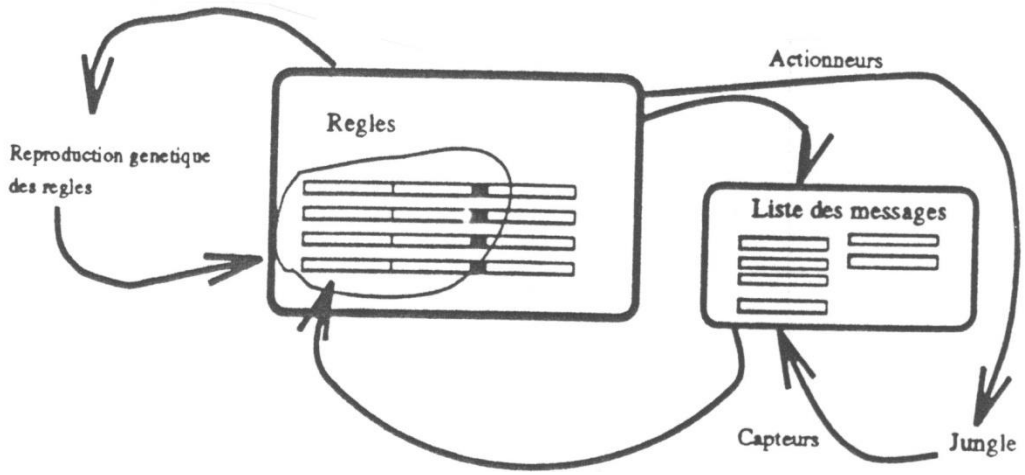
# Advantages of PG over AG

- Similar evolutionary approaches, but PG is not restricted to chromosomes of a priori fixed length.
- The elements of expressions can be of different complexities compared to the elements in the strings used in GAs.

- As in AG, the representation of the problem in terms of chromosomes is not obvious and can lead to false solutions if done poorly.

# Classifiers

learning by GA




- chromosome = classifier $\in \{ \mathbf{0} , \mathbf{1} , \mathbf{*} \}^{2l}$
  = rule = condition ( $\in \{ \mathbf{0} , \mathbf{1} , \mathbf{*} \}^{2l}$ ) + conclusion ( $\in \{ \mathbf{0} , \mathbf{1} \}^{l}$ )
- message storage area (on the blackboard)
- message ($\{ \mathbf{0} , \mathbf{1} \}^{l}$) = result of triggering a rule
- generation of new rules by an AG
- selection of the most useful classifiers and disappearance of the least interesting ones

# Principle

The condition of a classifier plays the role of a filter ( * = wildcard), if a message matches, the classifier is alerted .
An alerted classifier is likely to send a message if it is activated.
The choice of the activated classifier(s) is made by an auction system where the credit of a rule is proportional to its strength (i.e. its degree of adaptation, its importance).
When a message is used, the amount of the bids relating to its exploitation is credited to the rule which produced it.

- Regularly an AG produces new classifiers .
- The strength of a rule gives its degree of adaptation (for the GA).
- Only part of the rules are replaced and the selection most often uses a biased roulette.
- The newly introduced rules then participate in the following auctions, etc.

# Auctions ("bucket brigade")

| | | *t*=1 | | | | *t*=2 | | | | *t*=3 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n° | classifieur | Force | Msg | Al. | Ench. | Force | Msg | Al. | Ench. | Force | Msg | Al. | Ench. |
| 1 | 01**:0000 | 200 | | Env | 20 | 180 | 0000 | | | 220 | | | |
| 2 | 00*0:1100 | 200 | | | | 200 | | 1 | 20 | 180 | 1100 | | |
| 3 | 11**:1000 | 200 | | | | 200 | | | | 200 | | 2 | 20 |
| 4 | **00:0001 | 200 | | | | 200 | | 1 | 20 | 180 | 0001 | 2 | 18 |
| | Env | 20 | 0101 | | | 20 | | | | 20 | | | |

| | | *t*=4 | | | | *t*=5 | | | |
|---|---|---|---|---|---|---|---|---|---|
| n° | classifieur | Force | Msg | Al. | Ench. | Force | Msg | Al. | Ench. |
| 1 | 01**:0000 | 220 | | | | 220 | | | |
| 2 | 00*0:1100 | 218 | | | | 218 | | | |
| 3 | 11**:1000 | 180 | 1000 | | | 196 | | | |
| 4 | **00:0001 | 162 | 0001 | 3 | 16 | 156 | 0001 | | |
| | Env | 20 | | | | 20 | | | |

| *t*=5 |
|---|
| Force |
| 220 |
| 218 |
| 196 |
| 206 |
| 20 |

last active prime

50

# Applications

- Sorting (Kinnear), cache management (Paterson et al.), data compression (Nordin et al.), ...
- Image recognition (Robinson et al.), image classification (Zao), satellite image processing (Daïda), ...
- Time series prediction (Lee), decision tree generation (Koza), data mining (Raymer), …
- Classification of DNA segments (Handley), proteins (Koza et al.), ...
- Synthesis of electronic circuits (Koza),
- Robot motion planning (Faglia et al.), obstacle avoidance (Reynolds), robotic arm movement (Howley), …
- Modeling in mechanics (Schoenauer et al.), …

# REFERENCES


- *Genetic Programming FAQ,* 2002. http://www.cs.ucl.ac.uk/research/genprog/gp2faq/gp2faq.html
- Akbarzadeh, MR, Kumbla, K., Tunstel, E., Jarnshidi, M., "Soft Comuting for Autonomous Robotic Systems", *Computers and Electrical Engineering* , 2002: 26, pp. 5-32.
- Bojarczuk, CC, Lopes, HS, Freitas, AA, "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, JR 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, JR 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, JCF, Poli, R., "Efficient Evolution of Asymmetric 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, BT, Mühlenbein, H., "Genetic Programming of Minimal Neural Nets Using Occam's Razor", *Proc. Of 5th Int. Conf. On Genetic Algorithms,* 1993, pp. 342-349.

# Some pointers

- References

  - Genetics Programming I,II & III, 1992, 1994, 1998, John Koza *et al.*
  - Genetics Programming : an introduction, 1998, Banzhaf *et al.*
  - Advances in Genetics Programming I,II, 1994 Kinear , 1996, Angeline *et al.*
  - Machine Learning, Tom Mitchell, 1996
  - Artificial intelligence: A new Synthesis , Nils J. Nilsson, 1998

- Software

ftp://ftp.io.com/pub/genetic-programming/code/
koza-book-gp-implementation.lisp
http://garage.cp.msu.edu:software/lil-gp