MC906/MO416 Introduction to AI

Lecture 8

UNICAMP

Introduction to AI

Lecture 8 - Genetic Programming

Profa. Dra. Esther Luna Colombini
esther@ic.unicamp.br

Prof. Dr. Alexandre Simões
alexandre.simoes@unesp.br

LaRoCS – Laboratory of Robotics and Cognitive Systems

UNICAMP

Advanced Institute for Artificial Intelligence

Summary

- Genetic Programming
  - Definition
  - Properties
  - Representation
  - Applications
  - Example

Challenges

- What is Genetic Programming?
  - is an attempt to address one of the central issues in computer science:
    - "How can computers learn to solve problems without being explicitly programmed to do so?"
- Challenge:
  - Most branches of AI research are deterministic in nature and assume problems with some characteristics:
    - Correct
    - Consistent
    - With logical motivation
    - Accurate
    - Sortable
    - etc

GP: Nature

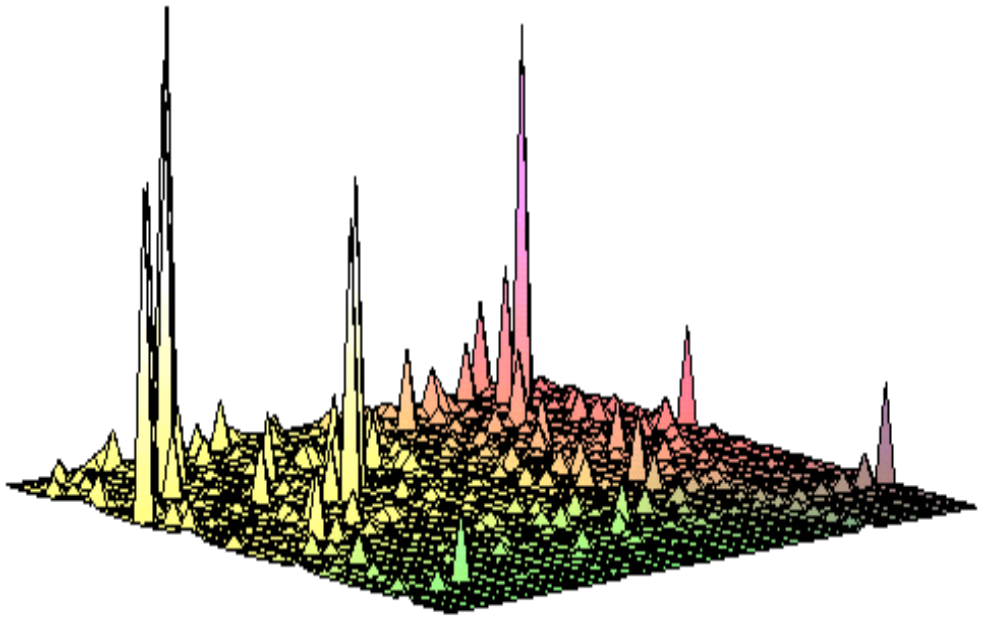
- PG
  - Handles correct and incorrect solutions
  - Encourages inconsistencies and contradictory approaches
  - Does not have a logical dynamic variability
  - It is predominantly probabilistic
  - Produces non-parsimonious solutions
  - Does not have a clearly defined termination criterion

GP: Definition

- "The evolution of computer programs using analogies with many of the mechanisms used by natural biological evolution." Leandro de Castro and Fernando Von Zuben
- It can be seen as an extension of genetic algorithms.
- A PG will be interpreted basically as a sequence of applications of functions to arguments
  - Functional paradigm
  - Individual = Program
- Original language of genetic programming: LISP
- Most used language in recent applications: C

GP: Goal

- Search for the best solution within the solutions space!
- Discover in the space of possible programs, a program that produces the output that satisfies the objective of the problem
- "If we are interested in computers solving problems without being explicitly programmed to do so, the structure must be a program." (Koza)



GP: Representation

- Data structures
  - List
  - Queue
  - Stack
  - Tree
  - Graph

GP: Problem

- For genetic programming, a problem is defined by:
  - Representation
  - Fitness function

PG: Representation

- Parse tree
  - most compilers translate the program into the parse tree
- The representation consists of functions and terminals, defining a language
- Functions and terminals: alphabet of the program to be induced
  - Terminal set: variables and program constants
  - Functions: addition, subtraction, division, ...
- **Computer programs in the defined language are individuals, capable of representation in tree-type data structures.**

GP: Type of functions

- Standard arithmetic functions
- Standard programming functions
- Standard mathematical functions
- Logical functions
- Domain specific roles

GP: Type of terminals

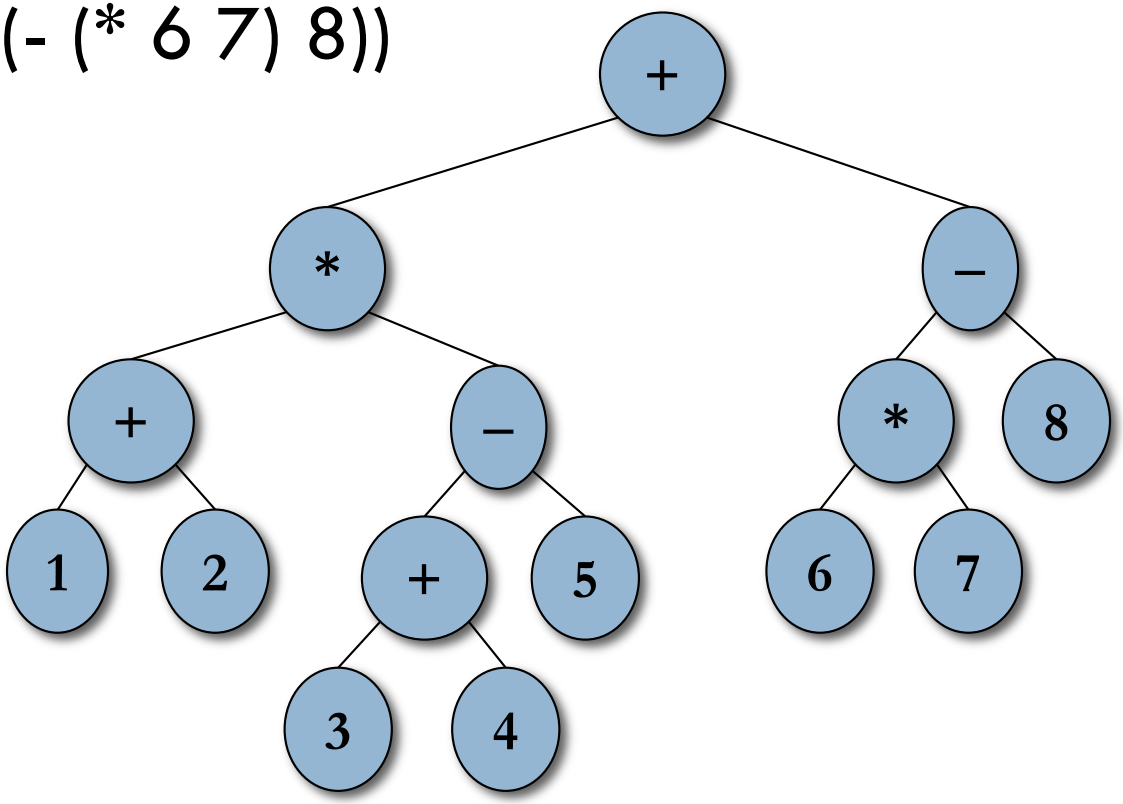
- Variables
- Constants
- Functions that take no arguments

GP: Examples of Representation

(+ (* (+ 1 2) (- (+ 3 4) 5)) (- (* 6 7) 8))

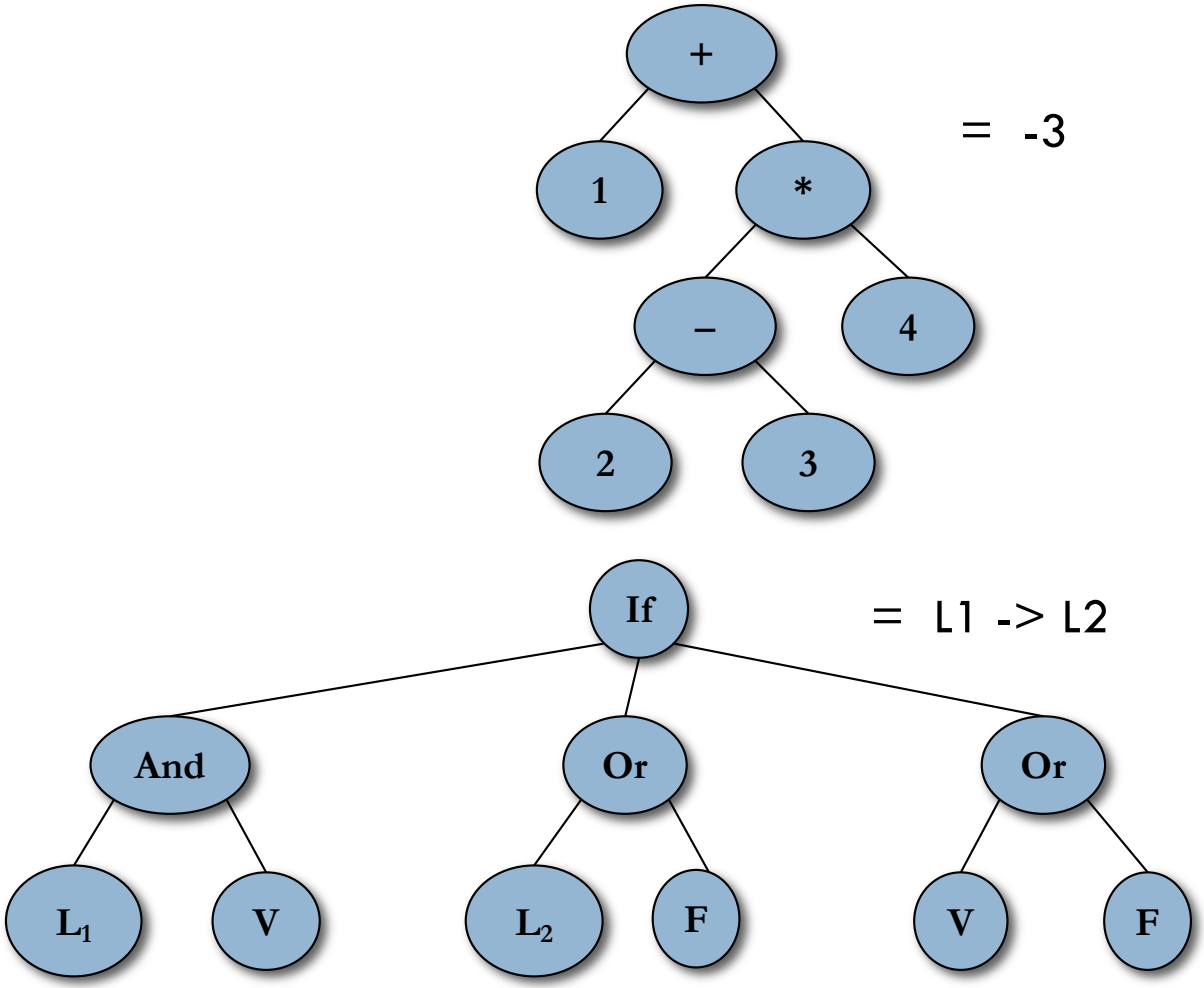
or

(+ (* (+ 1 2)
(- (+ 3 4) 5)) ou
(- (* 6 7) 8))



PG: Examples of Representation

- Functions:
  - $F = \{+, *, -, /\}$
- Terminals :
  - $T = \{0, 1, 2, 3, 4\}$
- Functions :
  - $F = \{If, And, Or\}$
- Terminals :
  - $T = \{L_1, L_2, V, F\}$
  - If (L_1&V) then
    - L_2 | F
  - Else
    - V

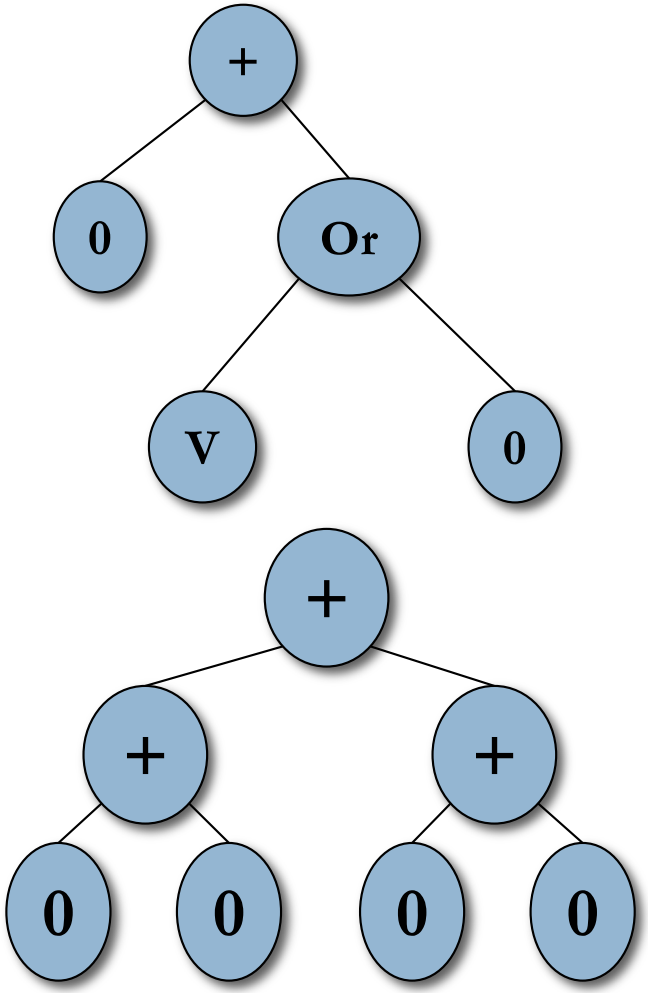


- Closing property (ensures tree viability)
  - Each function in the set F must accept, as its arguments, any value that can be returned by any function or terminal. This imposition ensures that any tree generated can be evaluated correctly
    - Example of Problem: division by zero

- Sufficiency property (to ensure convergence)
  - function sets F and that of T terminals must be able to represent a solution to the problem (Koza 1992). This implies that there must be strong evidence that some composition of functions and terminals can produce a solution

GP: Properties

- Closing
  - Functions: $F = \{+, Or\}$
  - Terminals: $T = \{V, 0\}$
- Sufficiency
  - Functions : $F = \{+\}$
  - Terminals: $T = \{0\}$



- **Partial credit:** the fitness function must measure not only the individual's absolute adaptability, but that related to more complete solutions
- Evaluation:
  - generally, each individual of the population is evaluated in several different situations and with more than one fitness function (software test), and their level of adaptation is extracted from the weighted sum of the levels of adaptation obtained for each situation

GP: Algorithm

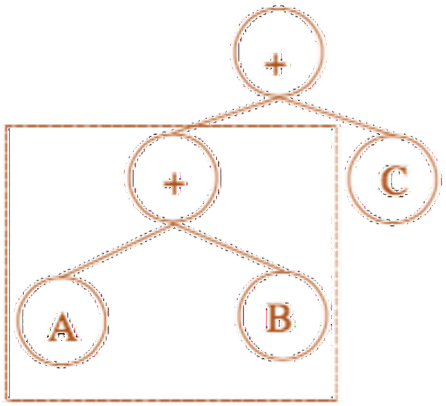
- Generate an initial population of random compositions of functions and terminals (computer programs)
- Perform the following steps iteratively until the stop criterion has been met:execute cada programa da população e atribua um valor de adaptação usando a função de *fitness*
  1. create a new population (generation) through:
     - recombination
     - mutation (absent in the original versions of algorithms for PG)
     - selection
  2. evaluate the result obtained in this new generation and test the stop criterion

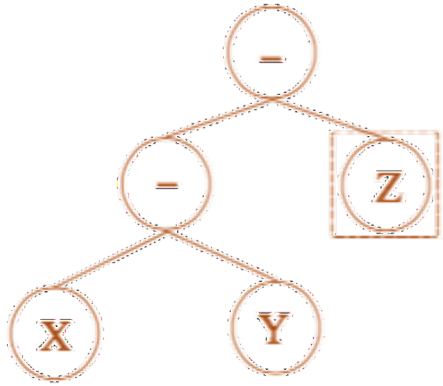
GP: Crossover

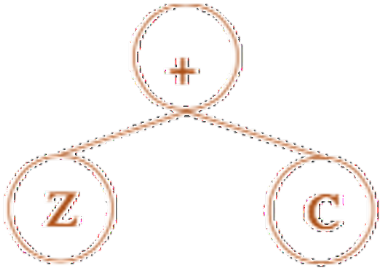
- Parent programs are generally of different size and shape
- The child programs are composed of sub-trees of the parent programs, and may be of different size and shape than those presented by the parent programs.
- The crossover point isolates a sub-tree from each parent expression, each of which may have arbitrary heights (within the limit established for the application)

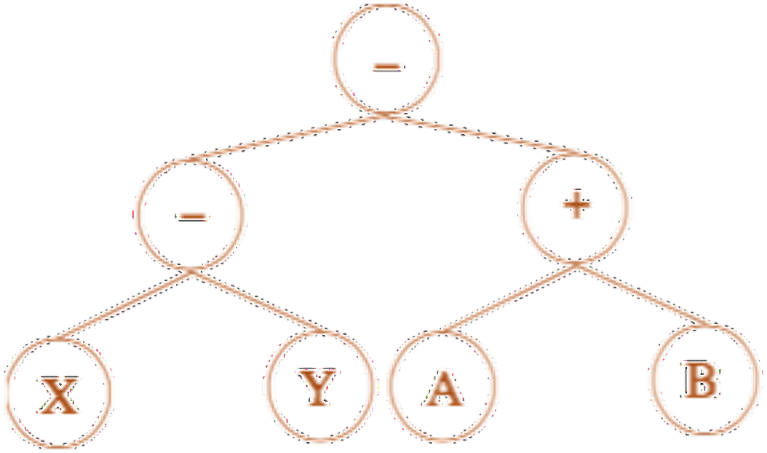
GP: Crossover

- Consider
  - **(+ (+ A B) C)**
  - **(- (- X Y) Z)**
- After crossover

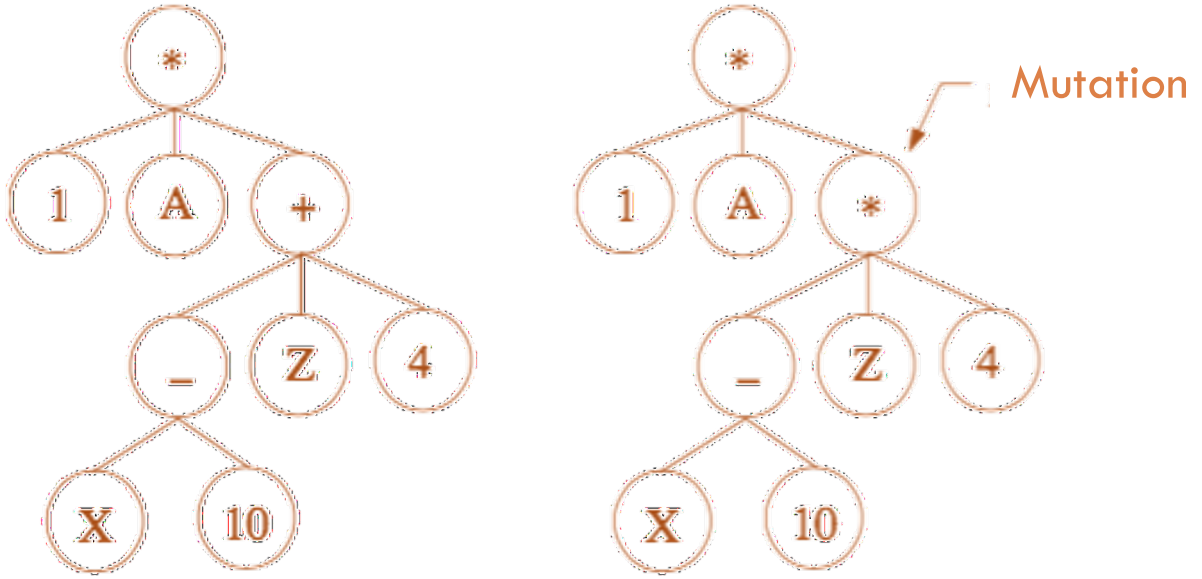








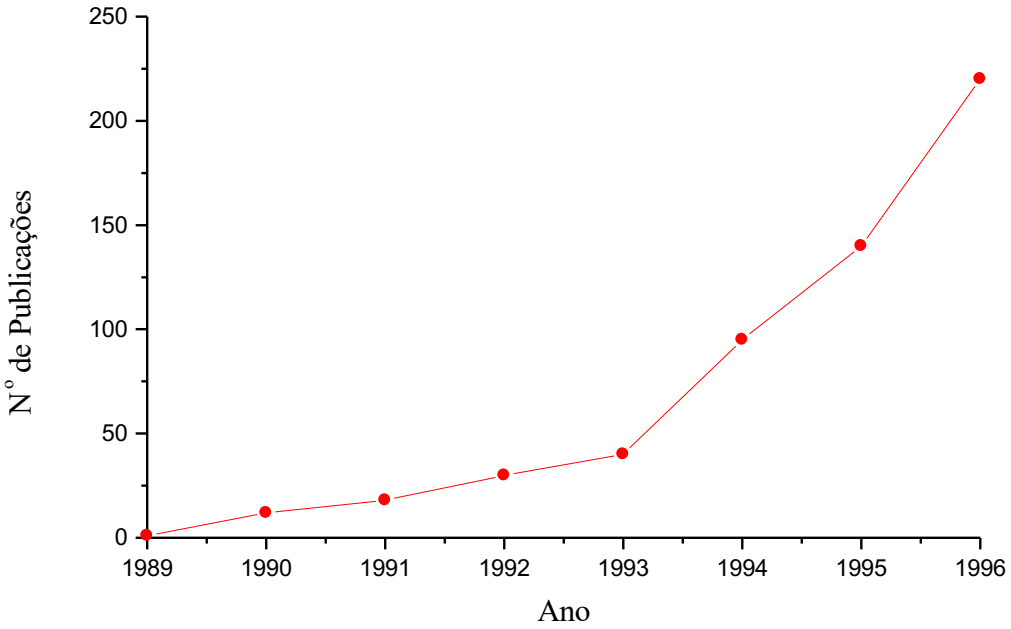
- The mutation can be useful, for example, in introducing changes in the roots of existing sub-trees, keeping all branches that leave there unchanged



GP: Distinction from AGs

- Absence of prior coding of the problem solution
- Each individual in the population (candidate for the solution) is a computer program
- The solution (or part of it) is obtained by executing the program and not by direct coding, as in genetic algorithms (the execution of the program should not be understood only in its strict sense)
- Hierarchically structured genetic material (tree structure), and not just neatly
- Genetic material of variable size during the evolutionary process (need to use size limiting mechanisms)
- Crossover with preservation of the correct syntax of genetic material
- The behavioral variability of the evolutionary process in PG is generally greater than in AGs.
  - It takes many generations to extract some kind of trend in relation to the results

GP: Applications

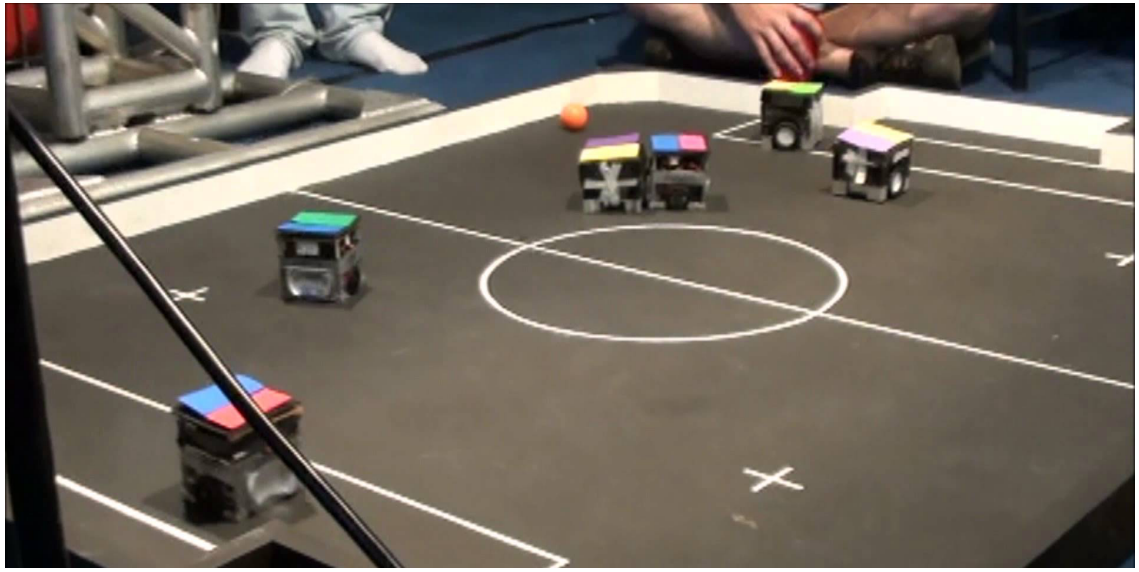


Domínio de Aplicação	Primeira Publicação
Algoritmos	1992
Arte	1993
Biotecnologia	1993
Gráficos de Computador	1991
Computação	1992
Controle (Geral)	1992
Controle (Processo)	1990
Controle (Robôs e Agentes)	1992
Controle (Espaçonave)	1996
Data Mining	1996
Engenharia elétrica	1994
Mercado Financeiro	1994
Sistemas Híbridos	1993
Processamento de Imagens	1993
Evolução Interativa	1991
Modelagem	1994
Linguagem Natural	1994
Otimização	1994
Reconhecimento de Padrões	1994
Processamento de Sinais	1992

GP: Wall-following robot

- Paper: MAIA, L. C., BiANCHI, R. A. C. . Usando Programação Genética para Evoluir Agentes Jogadores de Futebol de Robôs. Worcomp, 2000.
- Objective: to evolve the program of a wall-following robot where it must visit the largest number of squares adjacent to the wall within a simulation environment in a maximum number of steps.

- Pass each time the robot executes a terminal (act)
- Execution each time the complete robot tree is traversed once (there are several steps within an execution and you choose not to stop it before it ends)
- Maximum 200 steps



GP: Wall-following robot

- **Fitness:**
  - fitness = fit – (unfit / 500)
  - fit is the number of positions on the ideal path the robot has visited
  - unfit is the number of times he has deviated from the ideal (penalty) path or visited a cell more than once
- Maximum fitness is 780
- At each simulation, the individual starts in different positions and directions to prevent him from walking the path automatically without checking his environment.

GP: Wall-following robot

- Functions
  - PROGN3(3): performs three branches in a row;
  - PROGN2 (2): performs two branches in a row;
  - IFWALL (I): executes its left branch if no wall is detected by the robot (at most 1 step away) and otherwise executes its right branch.
- Terminals
  - WALKFRONT (F): makes the robot take a step forward
  - WALKBACK (B): makes robot take a step back
  - RIGHT (R): makes the robot turn right;
  - LEFT (L): makes the robot turn left.

GP: Wall-following robot

- Population (M) = 500
- Crossover Probability = 70%
- Simulation matrix = 200 x 200, with robot positioning in real numbers
- Limit of complexity or size of the solution (in the draw) = 1000
- Turning angle = 5o
- Number of executions per individual = 2

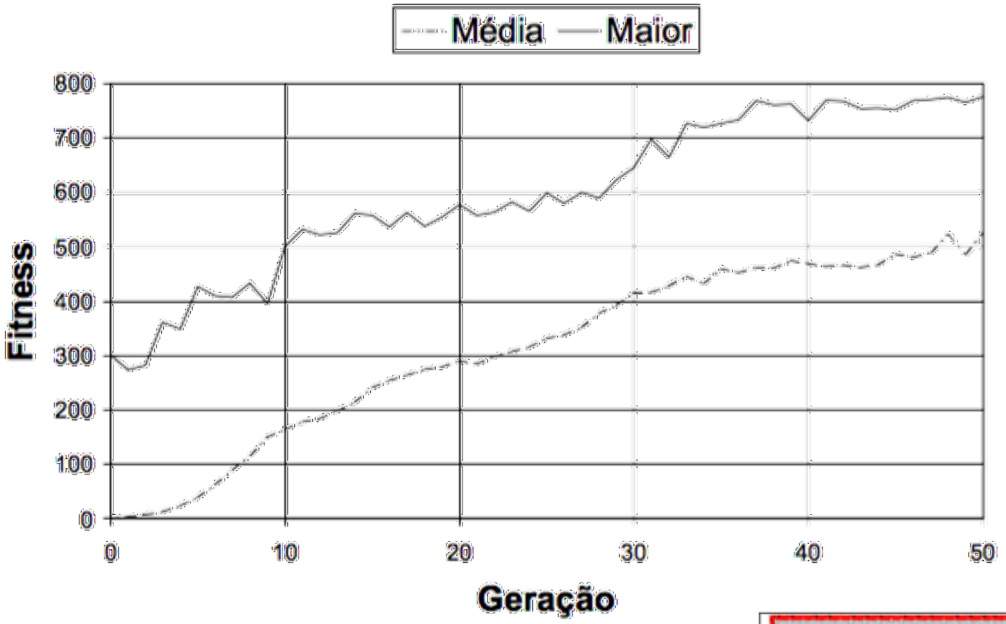
GP: Wall-following robot

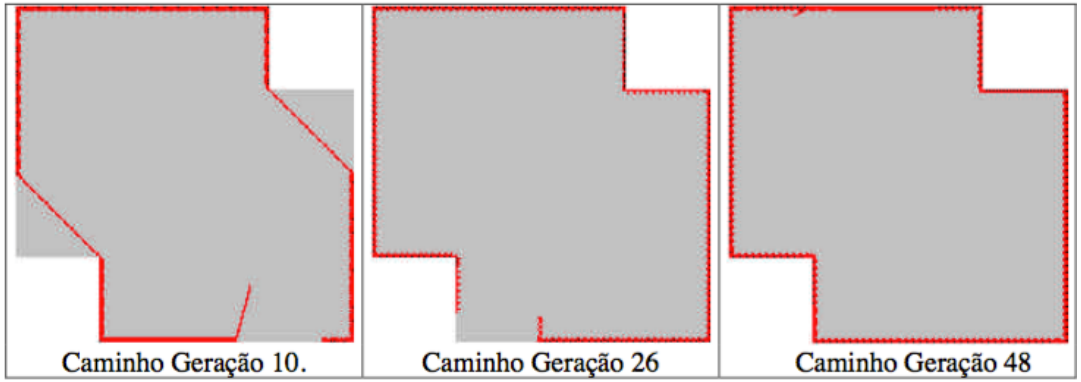
- Results:
  - Average fitness (Best agents for each simulation): 776
  - Average complexity (best agents): 290

  - PROGN3 = 3; PROGN2 = 2; IFWALL = I; WALKFRONT = F; WALKBACK = B; RIGHT = R; LEFT = L; ALIGN = A.

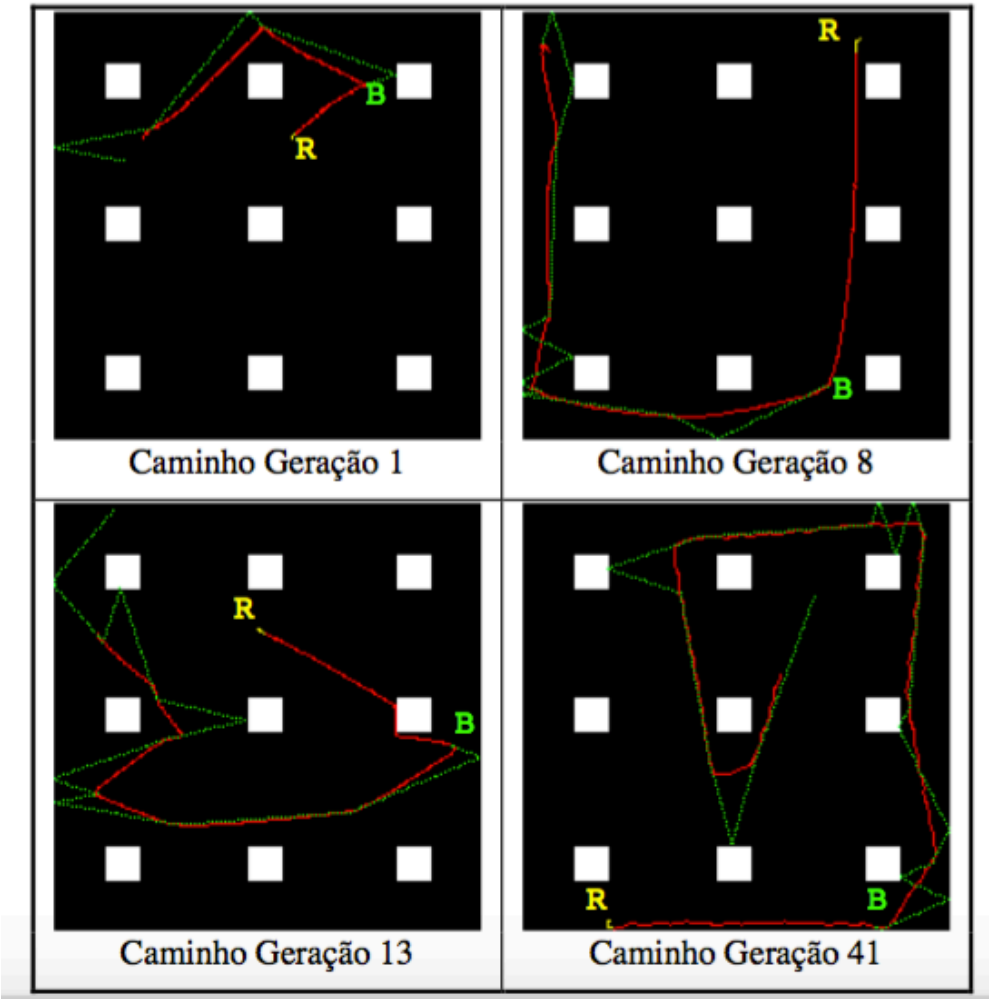
```
3IR22222FIF3B2II2FII2B2LI2B2B32F2LRRI22BL3IBIFIFBFB323L3B3BFB2F32RFF23IIB3FLIFRLR2L
3IILLILFILLRIF23B2II32RLIFRBB2RIIBB33IR223I3IILL2I2I33BBRBRBIRRL3IB32LRBRRBI2BRLRF
R2L3222FRIILLIFFRIFLRIF3BIBRLRFFR2FRRL3IILBILFIFLRR3IB2I32LIRR22ILRBRILBBBBLFLFB
FBLR2IBRRRRIILLIFFRFI33FLIFR2LIRBB3ILLFIR3BBR
```

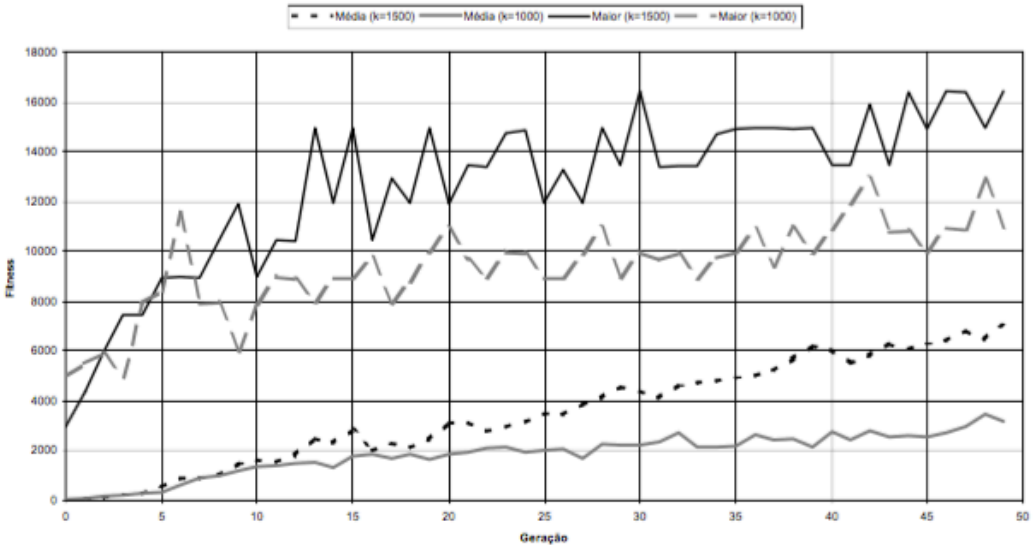
GP: Wall-following robot





GP: Ball-following robot





Average fitness (Best agents for each simulation): 16433

Average complexity (best agents): 535

Extra-class activities

Lecture 8

- Activities
  - Reading:
    - ZUBEN, F., CASTRO, L. IA707 – Tópico 12 Programação Genética (PG).
    - COPPIN, Ben. Inteligência Artificial, 2013. Capítulo 14.

References

Lecture 8

- ZUBEN, F., CASTRO, L. IA707 – Tópico 12 Programação Genética (PG).
- COLOMBINI, E. C, TONIDANDEL, F. Notas de Aula de IA.
- MAIA, L. C., BiANCHI, R. A. C. . Usando Programação Genética para Evoluir Agentes Jogadores de Futebol de Robôs. Worcomp, 2000.
  - COPPIN, Ben. Inteligência Artificial, 2013. Capítulo 14.