Genetic Programming

With some elements from: A.E. Eiben and J.E. Smith

Philippe Pasquier
Office 565 (floor 14)
pasquier@sfu.ca

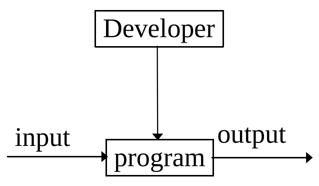
SFU

Philippe Pasquier, March 2008

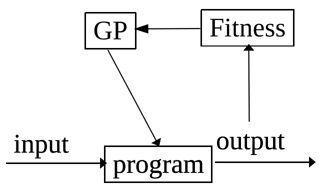
Introduction

- **Genetic programming (GP) extends genetic algorithm to non linear representation.**
- **Technique developped in the late 80's and ealry 90's:** Koza, John R. 1992. *Genetic* Programming: *On the Programming of Computers by Means of Natural Selection*. Cambridge: The MIT Press.
- **GP uses the principle of genetic algorithm but with programs as data**

Conventional programming



Genetic programming



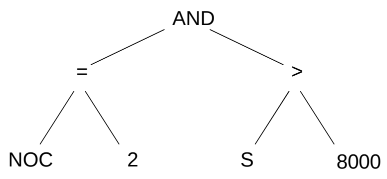
Introduction

- The key features of GP are:
  - Require big populations (thousands)
  - Require massive computational power
  - It is quite slow
  - But if it is used properly, it produces human-competitive results
  - GP deals with non-linear chromosome (tree, graph)
  - In this presentation, we will focus on trees

Tree-based representation

- Trees are special acyclic connected graphs made of:
  - Nodes
  - Each node can have a number of children
  - A node without children is called a terminal node or a leaf
  - One of the node is the root.
- A set of tree is called a forest
- Example:

  (NOC = 2) AND (S > 8000)



Tree-based representation

- Trees allow considering:
  - **Arithmetic formula** (and functional programming, e.g. caml)

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

  - **Logical formula** (and declarative programming, e.g. Prolog)

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

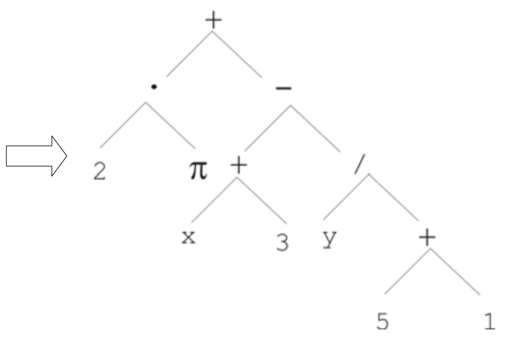
  - **Program (procedural programming, e.x. C)**

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

Tree-based representation

- Arithmetic formula

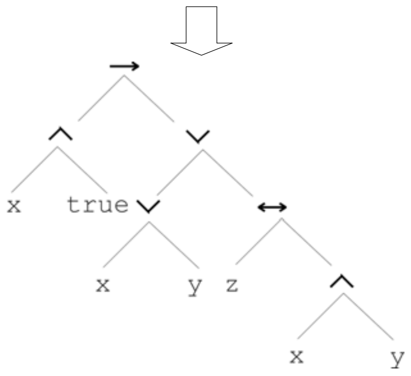
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$



Tree-based representation

- **Logical formula**

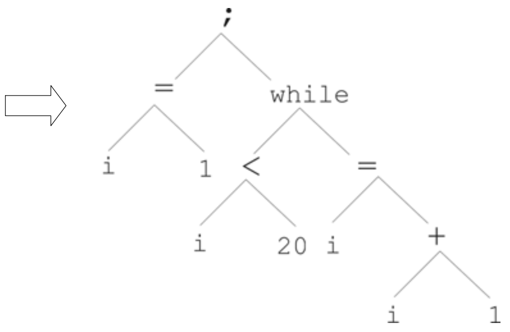
$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$



Tree-based representation

- **Program**

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



Tree-based representation

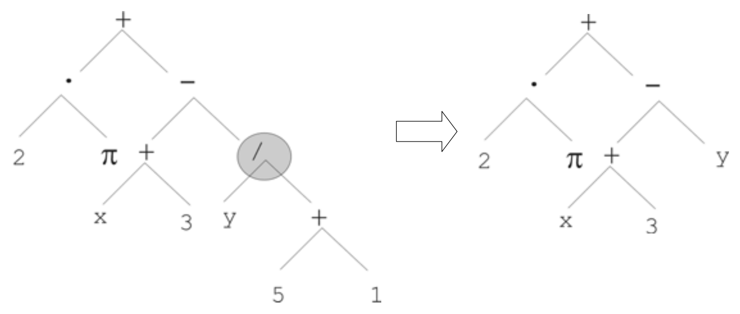
- Usually, we distinguish:
  - Terminal Set: Independent variables, Zero-argument functions, Random constants, ...
  - Function Set: Primitive functions that can be applied on the terminal set
- In GA, chromosomes are linear structures (bit strings, integer string, real-valued vectors, ...)
- Tree shaped chromosomes are non-linear structures
- In GA, the size of the chromosomes can be fixed or can vary
- Trees in GP may vary in depth and width

Offspring creation scheme

- GA and GP operate in similar ways:
  - GA scheme using crossover AND mutation sequentially (be it probabilistically)
  - GP scheme using crossover OR mutation (chosen probabilistically)
- Mutation rate is advise to be very low (0.05 at the maximum)
- Like for GA, one of the chalenge is to make sure that offsprings are well-formed programs

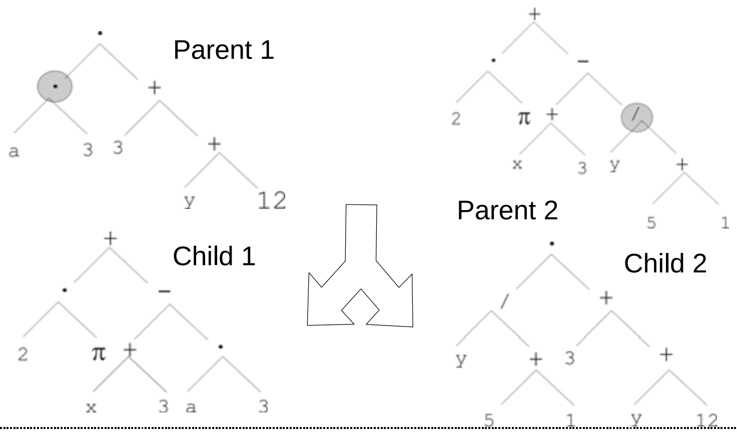
Mutation

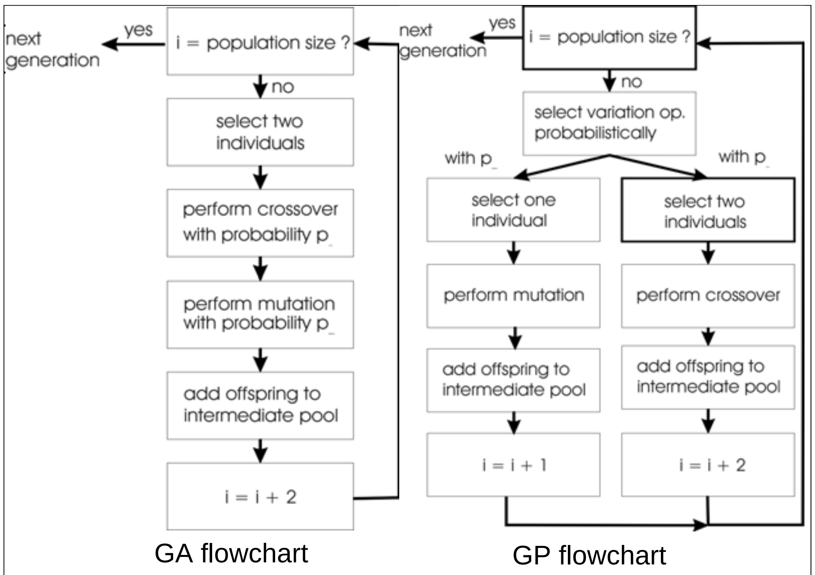
- **Most common mutation: replace randomly chosen subtree by randomly generated tree**



Crossover

- **Most common recombination: exchange two randomly chosen subtrees among the parents**





GA flowchart GP flowchart

Selection and Fitness function

- All the selection mechanism studied for GA can be applied here
- 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
- Fitness function:
  - Fitness functions are typically hard to define (e.g. binary nature of many solutions; how bad is a wrong answer?)
  - In the case of GP, the evaluation has to be done through a simulator (a system that can "run" the programs)

Performance of GP

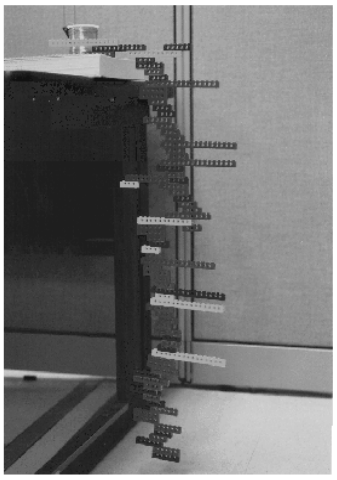
- Like for GA, the performance of GP depends crucially on the choice of representation and fitness function
- Bloat = "survival of the fattest", i.e., the tree sizes in the population are increasing over time
- Ongoing research and debate about the reasons
- Needs countermeasures, e.g.
  - Prohibiting variation operators that would deliver "too big" children
  - Parsimony pressure: penalty for being oversized

Examples

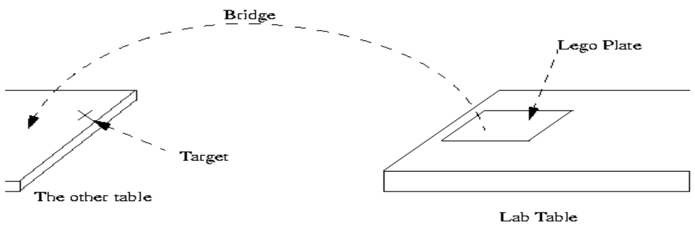
- There are dozens of problems for which the results of Genetic Programming were immediately better than the best known human counterpart
- Progression of qualitatively more substantial results produced by GP:
  - Toy problems
  - Human-competitive non-patent results
  - 20th-century patented inventions
  - 21st-century patented inventions
  - Patentable new inventions
- While it was initially designed for computer program, it is also applied to the computer evolution of buildable objects: electronic circuits, antennas, ...

Examples

- Evolutionary design
- Work by Pablo Funes and Jordan Pollack
- EvoCAD
- Scaffold evolved in 44000 generations
- Not optimized



Examples



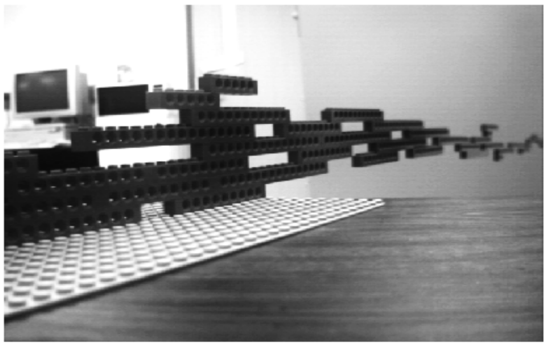
- Reaching a target point:
  - Bridges and Scaffolds
- Fitness Function:
  - Normalized distance to the target:

$$Nd(S,T) = 1 - \frac{d(S,T)}{d(0,T)}$$

Where S is the structure, T is the target point and d is the Euclidean distance

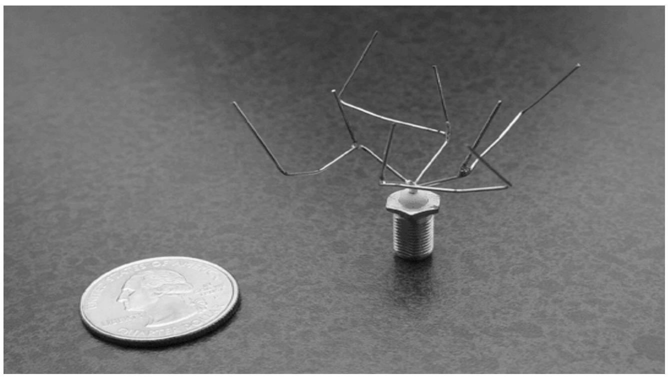
Examples

- The target fitness was reached after 133,000 generations



Examples

- NASA antennas: used on satelittes since 2004



Models of evolution and learning

- Lamarckian evolution:
  - Lamark (late 19th century) developped the following conjecture:
    - The experience of an organism affect the genetic makeup of their offsprings
  - Despite that current scientific evidence contradict Lamark's model, it has proven to be usefull in some case to extend/refine GA or GP and improve their effectivness (this is artificial life in the sens of “life as it could be”)

Models of evolution and learning

- Other mechanisms by which individual learning can affect the course of evolution have been proposed
- Baldwin effect:
  - Individual are evolving in changing environment so their should be evolutionary pressure to favor individual that can learn and adapt suring their lifetime
  - Individual who can learn many traits will rely less on “hard-wired” traits
  - That will have an indirect accelerating effect on evolutionnary adaptation of the whole population

Conclusion

- 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)
- The solution: evolutionary computing as an invention machine
- Genetic programming is a variant of genetic algorithms in which the hypothesis being manipulated are computer programs (rather then bit strings)
- Is GP:
  - The art of evolving computer programs ?
  - Means to automate programming of computers?
  - GA with another representation?

?

“Today, the theory of evolution is an accepted fact for everyone but a fundamentalist minority, whose objections are based not on reasoning but on doctrinaire adherence to religious principles”

James D. Watson