Why you should be interested in Genetic Programming

David Andre
dandre@cs.cmu.edu
dandre@cs.berkeley.edu
Robot Learning Talk, Oct 9, 1997

Outline:

1) What we are looking for?

2) Details of GP

3) What has GP done for us lately?

4) Evaluation -- have we found what we are looking for?

Machine Learning:

"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?"

Arthur Samuel, 1954

"If we could understand how to program [computers] to learn -- to automatically improve with experience -- the impact would be dramatic. The practice of computer programming would be revolutionized as many tedious hand-coding tasks would be replaced by automatic learning methods."

Tom Mitchell, 1996

Goal is program induction:

- Has to be search because we don't know how to code the programs (otherwise humans can code)
- Two extremes:
- **Knowledge based** - hard code in enough knowledge to reason out what the best program is (with little or no search)
- **Random Search** -- generate random programs and test them for their worthiness through simulation

- We want something between these situations.
- So, what are the characteristics of the system that we want?

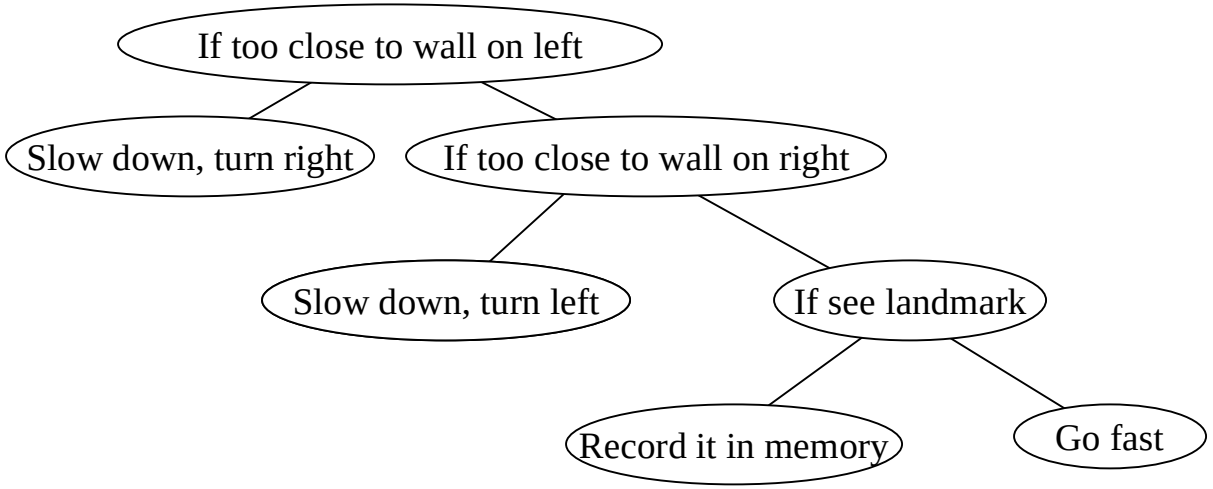
  - learning programs (general structure -- applicable to many probs)
  - insertion of domain knowledge
  - evaluation through simulation (or a mixture of simulation and real world experience, but a minimum of real world time
  - system should 'learn' about problem domain through search process
    - later "solutions" and improvements should be based on that knowledge
  - we want to be doing beam search of some sort, helps to avoid minima
  - not always greedy
  - can use hierarchy of problem space to advantage

Genetic Programming

- Artificial evolution of computer programs

- **1) Create random population of tree structured computer programs**
- **2) Evaluate fitness of each individual in population**
- **3) Create new population from old population, using a variety of operators, including crossover and mutation**
- **4) Go to 2 unless done**

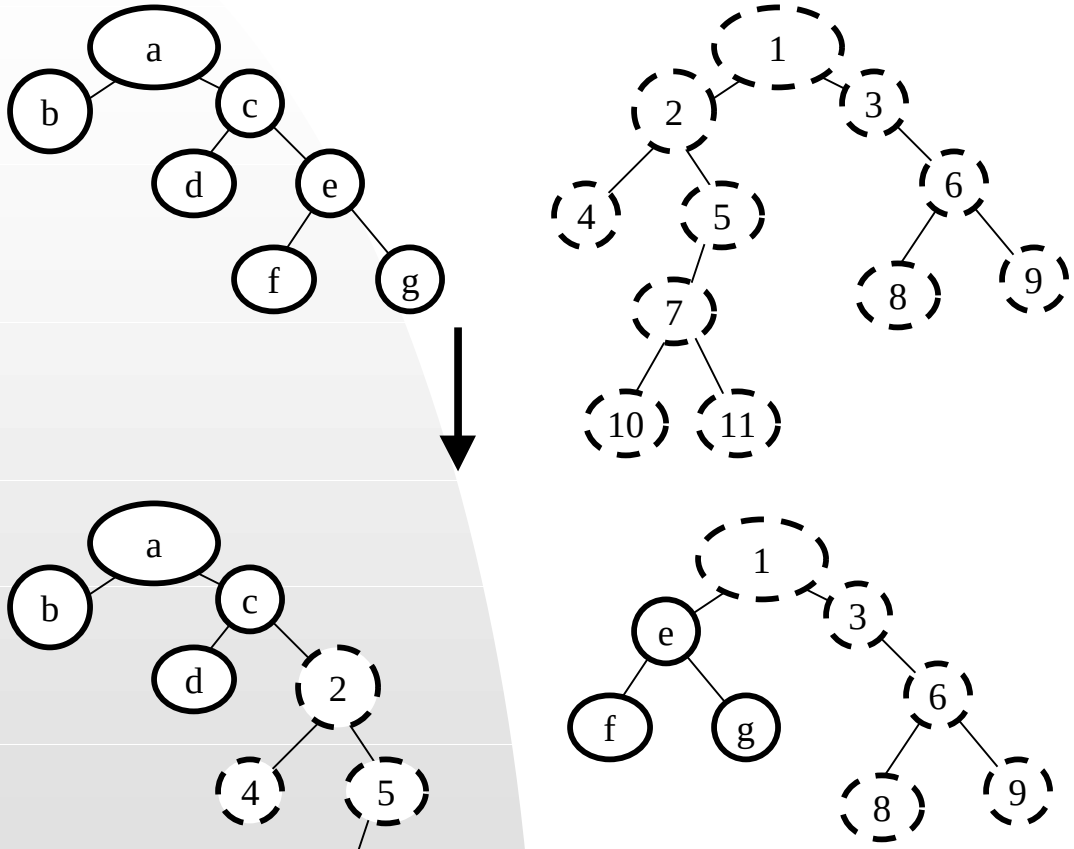
Basic Tree Structure



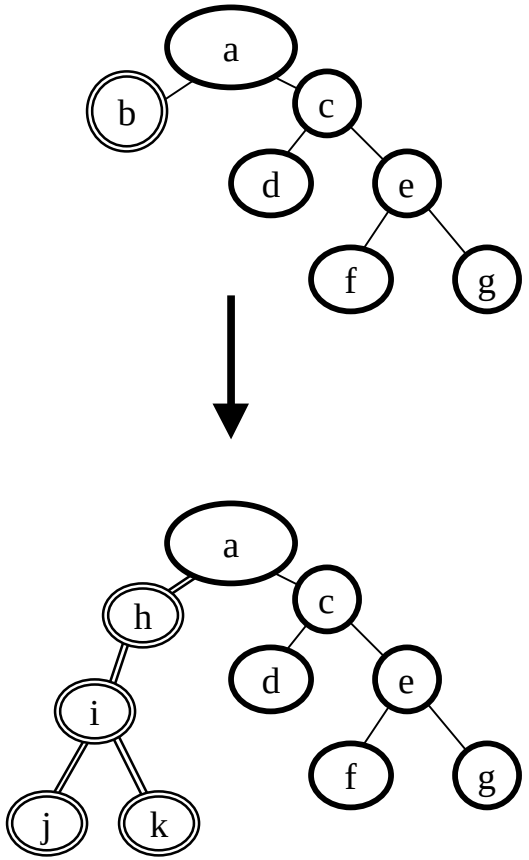
- We can have arbitrary functions, such as arithmetic, memory, loops, side-effecting functions (move, etc), macros, subroutines, etc.
- Hope is that subtrees have some locality in behavior space.
- Evolving programs may be embedded within arbitrary test harnesses.
- Invented by Koza, 1992 -- based on Holland's Genetic Algorithm

Operators

Crossover



Mutation



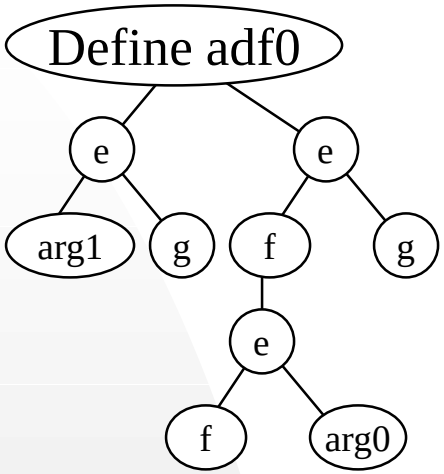
Simple Examples

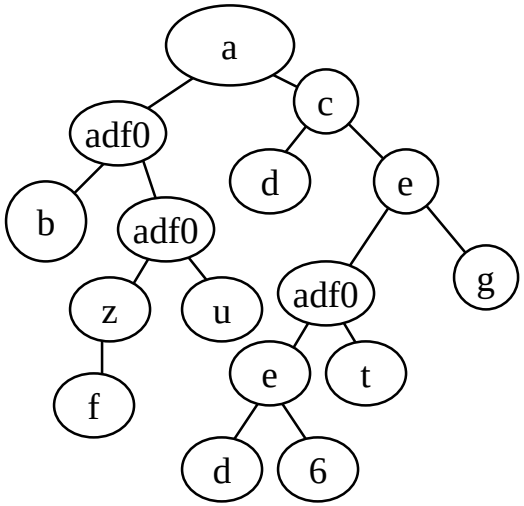
- Symbolic regression:
  - +,-,*,%, sin, cos, x, randconst, etc.
  - discovers size and shape of solution
  - is same as prediction, system id, etc
- Control
  - Wall follower, box pusher (etc)
  - Video games (tetris, pac man)
- Classification
  - OCR, image recognition
  - can use moving feature windows
  - Directed attention
- Many more --see Koza, 1992

For each problem:

- Choose functions and terminals
  - These are the programmatic ingredients
- Design a fitness function
- Decide the run parameters
- Decide which program is to be interpreted as the result
- Decide the architecture of the program
- Decide any problem specific items (memory initializations, intitial circuit structure, etc).

Parameterized Subroutines





- Adfs co-evolve with a main branch that calls them
- Adfs can be embedded, but no recursion is typically allowed
- Almost always speed evolution

Problems where GP has done well

- Wide variety of simple problems
- Image processing
- Molecular Biology problems
  - Transmembrane detection
  - Motif identification problem
  - Secondary structure prediction
- Control
- Cellular automata
- Evolution of neural networks
  - Robotic controllers for pole balancing, six legged beast
- Evolution of analog electrical circuits
  - Simple filters
  - Complex hierarchical filters
  - Computational circuits (sine, square root, multiply)
  - Op Amps
- Theorem proving
- Evolvable hardware
  - Learned robotic controller for Khepera in real world

Evaluation of GP

- Is it what we are looking for?
  - Evolving **programs**
  - Allows solution of wide range of problems by changing function set
  - Tree based representation is easy to manipulate, and is good for locality of changes (even when the trees represent circuits)
  - Pre-existing knowledge can be coded in function sets (useful tools) or in the fitness function (identification of "bad ideas")
  - Beam search
    - By retaining a large population of sucessfully reproducing results, a body of implicit knowledge is kept about the search process.
    - Population helps to avoid minima, and restricts space of future progeny
- Crossover
  - Crossover combines aspects of two relatively successful programs
  - Counts on fact that sucessful individuals will have successful subtrees
  - Is thus better than inserting random subtrees
- Not always greedy
- Can exploit hierarchy, regularity in the problem space with ADFs.

On the other hand...

- Not enough theory,
  - why does crossover help?
  - When will GP solve the problem?
- Not enough comparison to other techniques
  - Plenty to random search (on nearly all problems GP is better)
  - Some to hill climbing (on hard problems GP is better)
  - very little to parallel simulated annealing, other techniques
- Would like "guesses" at new solutions to be less random
- Too much dependence on biological metaphor (intron story)
- Why only crossover and mutation? Many other potentially useful operators (bias).
- Requires large populations and many evaluations

Current directions

- Solve real-world hard problems
- Search for explaining why crossover works,
- Search for theory as to when GP will solve
- Explore rate of exploration vs. rate of convergence issue
- Create situations to "guide" crossover, ideally we want to make
  - most "rational" choices
  - ie, we want to create a new program that has the highest probability of being better and a high probability of yielding us new information

Impact for Robot Learning

- Very general method for problem solving.
- Low ramp-up time
- Requires simulation or not too many fitness cases
- Some variants (EP, ES, etc) may well be better for RobL
- Good when approximate solutions are satisfactory
- Programs are difficult to understand -- but strong typing helps
- Can be used in collaboration -- can use human routines as primitives, then humans can modify output of GP, etc