Evolutionary Algorithms: Genetic Programming

Adrian Łańcucki

Instytut Informatyki
Uniwersytet Wrocławski

December 9, 2013

Intuition

- Genetic Programming
- not to confuse with any other kind of buzzword-Programming
- in this case, it is the computer (specifically the algorithm) that does the programming
- it happens through an evolutionary process
- GP is not that powerfull as you wish it was
- yet powerfull enough for interesting, real-world applications

- 1964, Fogel - discovering DFA with EA
- 1981, R. Forsyth - BEAGLE
- 1985, N. Cramer - GA applied to special programming languages,
- 1987, Fujiki and Dickinson - GA on a subset of LISP
- 1987, simple FORTRAN-based ideas, rarely compiled
- 1996, Nordin and Banzhaf - Linear GP
- 1996, Poli - Parallel Distributed GP
- 1997, Miller - Cartesian GP
- ...

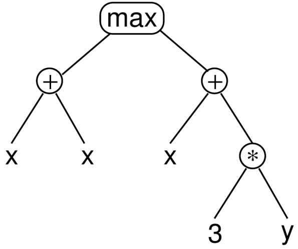
John Koza

- 1972 - PhD in Computer Science (University of Michigan)
- - 1987 - Scientific Games Corporation
- 1988 - MIT
- 1992 - Genetic programming
- 1995 - first recreated patent (low-pass filter-a-circuit)

Basic GP Algorithm

```
P ← InitialPopulation(n);
while not TerminationCondition(P) do
    P' ← {};
    EvaluateIndividuals(P);
    while |P'| ≠ |P| do
        p1 ← TournamentSelection(P);
        p2 ← TournamentSelection(P);
        (o1, o2) ← Crossover(p1, p2);
        o1 ← Mutation(o1);
        o2 ← Mutation(o2);
        P' ← P' ∪ {o1, o2};
    end
    P ← P';
end
```

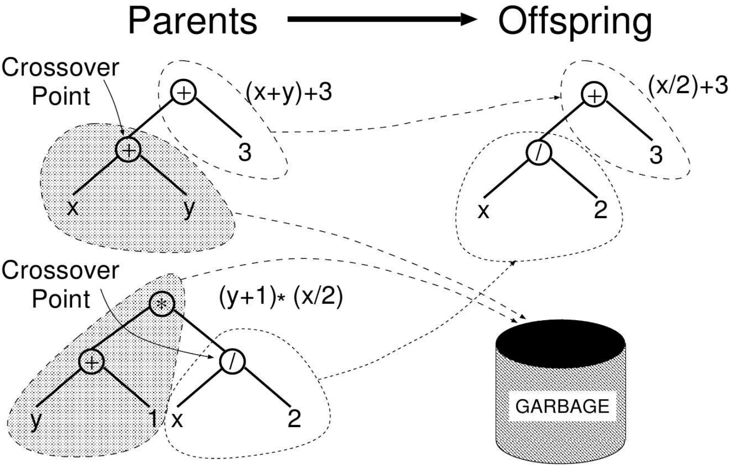
Representation



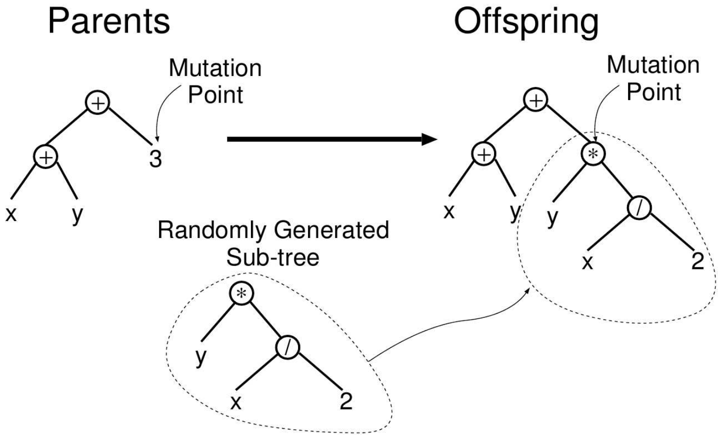
$$max(x + x, x + (3 * y))$$

- what can we tell about the search space?
- usually, we would like to constrain the size of the chromosome
- this is a valid program (or an expression)
- bottom-up evaluation

Crossover Operator

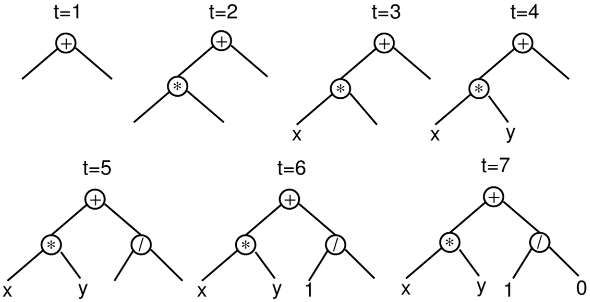


Mutation Operator



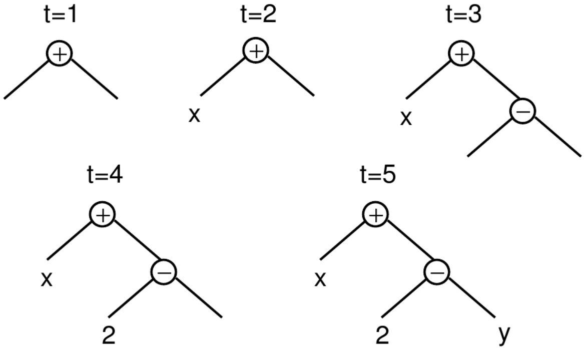
Population Initialization

- `full` method



Population Initialization

- `grow` method

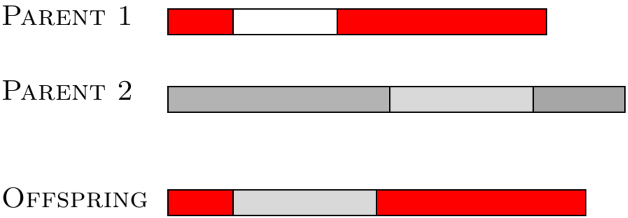


- in practice: `ramped-half-in-half` method

Demo

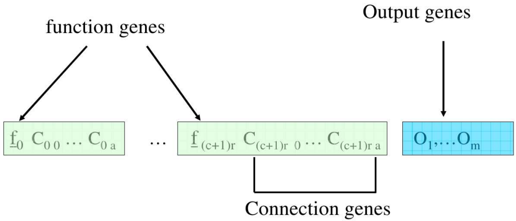
Linear GP

- tree structures are not that close to "real" coding
- source code is written as consecutive steps, so let chromosomes be linear
- is software really that fragile?
- linear crossover, mutation

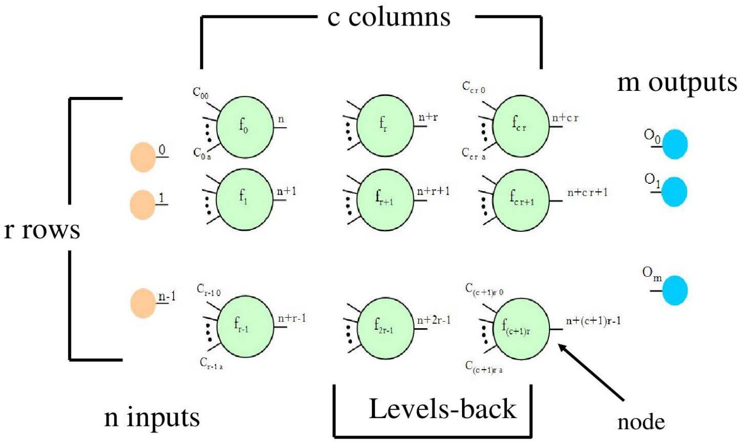


Cartesian GP

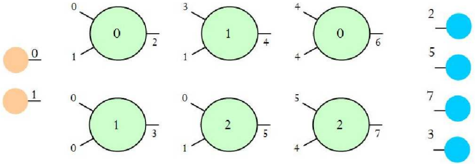
- Miller (1999)
- encodes graph structures
- genotype is (sort of) a squashed incidence matrix



Cartesian GP (cont'd)



Cartesian GP (cont'd)

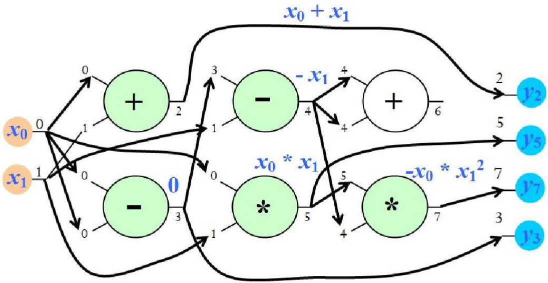


Function gene (address)	Action
0	Add
1	Subtract
2	Multiply
3	Divide (protected)

Genotype

0 0 1 1 0 0 1 3 1 2 0 1 0 4 4 2 5 4 2 5 7 3

Cartesian GP (cont'd)



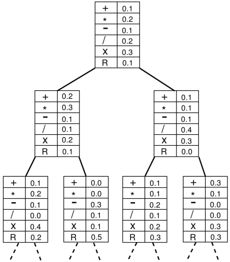
$$y_2 = x_0 + x_1$$
$$y_5 = x_0 * x_1$$
$$y_7 = -x_0 * x_1^2$$
$$y_3 = 0$$

Cartesian GP (cont'd)

Properties

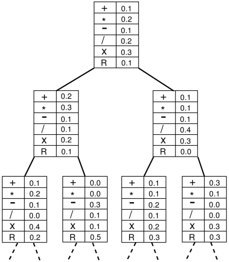
- mutation as the main search operator
- new crossover proposed in 2007 (Clegg, Walker and Miller), provides faster convergence
- some connections might be ignored (due to different function arities)
- some nodes might be ignored (not reachable from input nodes)
- thus genotype to phenotype mapping is many-to-one
- CGP typically uses a (1+4) evolutionary strategy
- easily encodes computer programs, electronic circuits, neural networks, math equations, ...

Probabilistic GP - PIPE



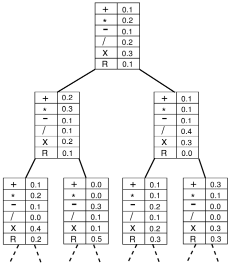
- Salustowicz and Schmidhuber, 1997
- only one Probabilistic Prototype Tree (PPT) is maintained
- PPT is a complete n-ary tree with infinitely many nodes
- each node has
  - P_T - probability of selecting terminal node
  - $\vec{P}_j$ - instruction set probability vector
  - R_j - constant picked uniformly from $[0; 1)$
- subtrees are pruned when terminals are highly probable, e.g. $P_T > 0.99999$
- PPT might be grown on-demand; usually should be twice as large as individuals

Probabilistic GP - PIPE (cont'd)



GBL();
while *not TerminationCondition*() **do**
    if *Rand*$(0,1) > P_{el}$ **then**
        EL();
    else
        GBL();
    end
end

Probabilistic GP - PIPE (cont'd)



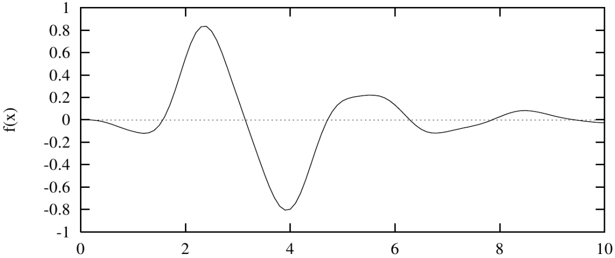
- EL - Elitist Learning (only from $Prog^{el}$)
- GBL - Generation-Based Learning
  - creation, evaluation and learning from population
  - mutation and prunning of PPT

$$P(\textsc{Prog}_b) = \prod_{j:N_j \text{ used to generate } \textsc{Prog}_b} P_j(I_j(\textsc{Prog}_b))$$

$$P_{TARGET} = P(\textsc{Prog}_b) + (1 - P(\textsc{Prog}_b)) \cdot lr \cdot \frac{\varepsilon + FIT(\textsc{Prog}^{el})}{\varepsilon + FIT(\textsc{Prog}_b)}$$

$$\text{REPEAT UNTIL } \; P(\textsc{Prog}_b) \geq P_{TARGET} :$$
$$P_j(I_j(\textsc{Prog}_b)) := P_j(I_j(\textsc{Prog}_b)) + c^{lr} \cdot lr \cdot (1 - P_j(I_j(\textsc{Prog}_b)))$$

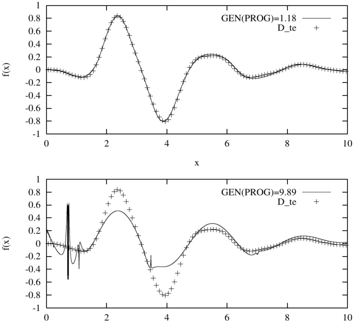
Probabilistic GP - PIPE - function regression example



$$f(x) = x^3 \cdot e^{-x} \cdot cos(x) \cdot sin(x) \cdot (sin^2(x) \cdot cos(x) - 1)$$

- $F = \{+, -, *, \%, sin, cos, exp, rlog\}$
- $T = \{x, R\}$
- PE = 100,000, FITs = 0.001, e = 0.000001, PeI =0.01, PS=10, lr=0.01, PM =0.4, mr=0.4, TR =0.3, TP =0.999999

Probabilistic GP - PIPE - function regression example



Best one (upper-left corner) found after 99390 evaluations.

Probabilistic GP - PIPE - function regression example

Best individual:

```
(sin(((x-((cos(((sin((sin(cos((rlog(sin(0.350466))*(((cos(
sin((cos((x-((rlog(cos(((0.359722+cos(x))+(x-0.082538))))*(x-
(0.039232-((x%0.440611)%0.499641))))*0.025812))-(0.914140*
(x*(0.506207%0.379995))))))*(x-((x%sin(rlog(0.334052))+rlog(((
x+x)*x)))))%exp(exp((0.743179-0.128703))))+x))))%(((0.507077*
((exp((x-x))-((cos(sin((x-(cos((0.915233%x))-exp(0.709387)))))-
0.492354)%0.840741))%cos((x*0.981004))))*((x%cos(x))*(0.091520*
(0.112682+sin(sin(x))))))+x)))%x)%(sin((((cos(sin(rlog((exp(x)%
cos(0.712427)))))+0.933998)%0.609029)-cos(0.936381)))%(((cos(
0.790039)-(x-0.069650))*sin(x))-x))))+sin(exp(rlog(x))))-((
sin(0.375208)*(exp(rlog(exp(0.697598)))%cos((cos(0.585192)-
0.095603))))+(((0.395458-(0.282354*sin(0.822447)))%(0.533448%
(0.785156*0.918876)))*cos((x-(0.639372%0.524799)))))))-sin((
sin(sin((sin(sin(sin(x)))%sin(cos(0.287498)))))+x))))*(cos(((
0.482642+((0.183318*(0.338145+0.069478))*cos((x+0.496698))))+
```

Probabilistic GP - more recent algorithms

eCGP (Sastry and Goldberg, 2003)

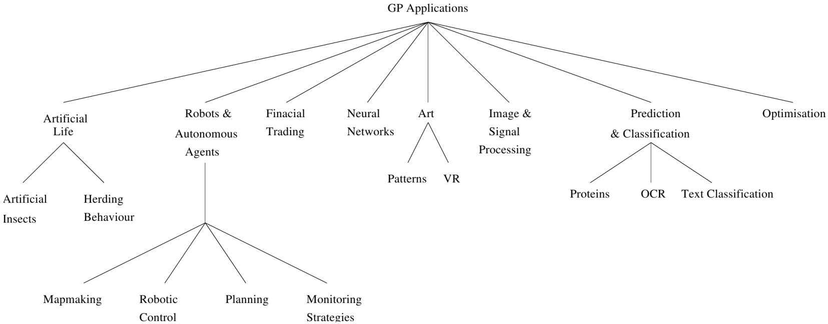
- Extended Compact Genetic Programming
- combines PIPE with eCGA

EDP (Yanai and Iba, 2003)

- Estimation of Distribution Programming
- ppb is estimated through a Bayesian Network
- no classical crossover nor mutation

N-gram GP (Poli and McPhee, 2008)

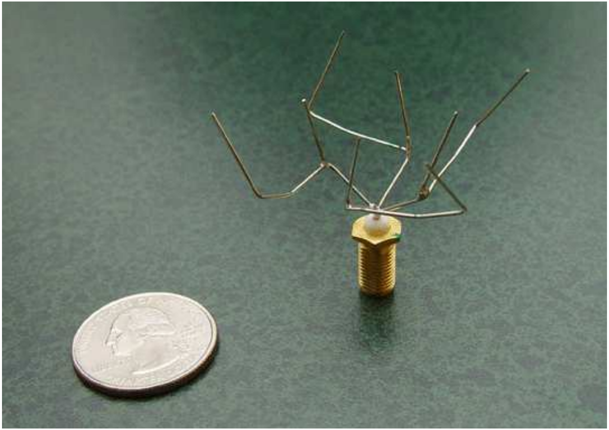
- allows evolution of linear programs
- better local search


GP Applications
Artificial Life
Artificial Insects
Herding Behaviour
Robots & Autonomous Agents
Mapmaking
Robotic Control
Planning
Monitoring Strategies
Finacial Trading
Neural Networks
Art
Patterns
VR
Image & Signal Processing
Prediction & Classification
Proteins
OCR
Text Classification
Optimisation

Popular Benchmarks

- Boolean Functions
- Classification
- Predictive Modelling
- Path-finding and Planning
- Control Systems
- Game Playing
- Dynamic Optimization
- Traditional Programming
- Constructed Problems

Sample Applications



ST5 antenna evolved by Jason Lohn and his group on Evolvable Systems at NASA Ames

Sample Applications (cont'd)

1000-Pentium Beowulf-Style Cluster Computer (July 29, 1999)
Genetic Programming Inc.

Sample Applications (cont'd)

A Genetic Programming Approach to Automated Software Repair

- Stephanie Forrest, Claire Le Goues, ThanhVu Nguyen, Westley Weimer (2009)
- automatic repair of legacy C code
- programs are loaded as abstract syntax trees
- genetic operators only local to a particular execution path
- positive and negative test cases serve for fitness assessment
- code bloat reduction with heuristics

Sample Applications (cont'd)

Microsoft Zune bug (Dec 31, 2008 freeze)

```
void zunebug(int days) {
  int year = 1980;
  while (days > 365)  {
    if (isLeapYear(year)){
      if (days > 366)  {
        days -= 366;
        year += 1;
      }
      else {
      }
    }
    else {
      days -= 365;
      year += 1;
    }
  }
  printf("current year is %d\n", year);
}
```

```
void zunebug_repair(int days) {
  int year = 1980;
  while (days > 365) {
    if (isLeapYear(year)){
      if (days > 366) {
        // days -= 366; // repair deletes
        year += 1;
      }
      else {
      }
      days -= 366;      // repair inserts
    } else {
      days -= 365;
      year += 1;
    }
  }
  printf("current year is %d\n", year);
}
```

Problems of GP - Bloat

- expressions grow indefinitely in size
- slow evaluation, memory overruns

Possible remedies:

- constraining tree size
- penalty for large trees
- bloat-aware operators (size-fair crossover)

Problems of GP - High Evaluation Cost

- Caching of outcomes of subprograms
- Parallel execution of programs on particular fitness cases
- Bloat prevention methods
- JIT of individuals
- Linear Programming might be translated directly to machine code

- Riccardo Poli, Bill Langdon, Nic McPhee *A Field Guide to Genetic Programming*,
- John Koza *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press
- Rafał Sałustowicz *Probabilistic Incremental Program Evolution*, PhD thesis, Berlin 2003
- Bill Langdon, Adil Qureshi *Genetic Programming – Computers using "Natural Selection" to generate programs*,
- Stephanie Forrest, Claire Le Goues, Westley Weimer, ThanhVu Nguyen *Genetic Programming Approach to Automated Software Repair*,