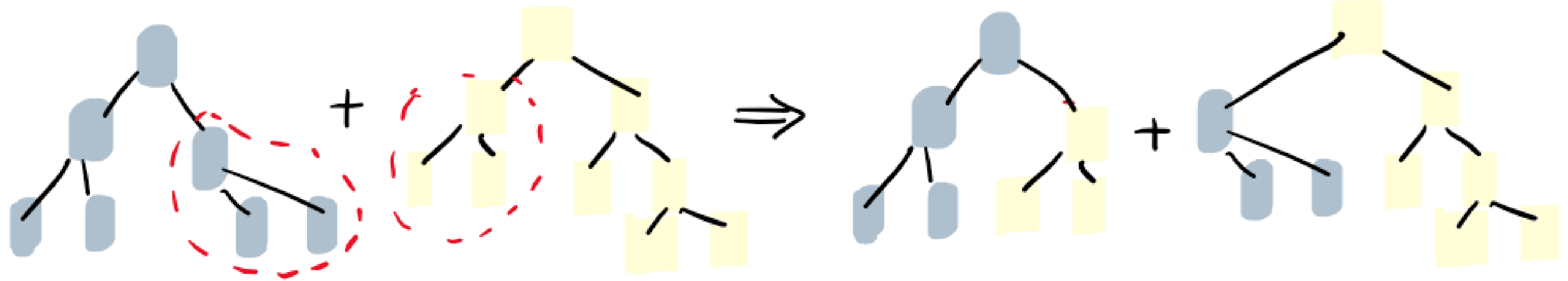
Data Driven Engineering II: Advaced Topics

Genetic programming:
towards data driven control

Institute of Thermal Turbomachinery
Prof. Dr.-Ing. Hans-Jörg Bauer



Convergence

* Max. generation
* Elapsed time
* Track fitness → Best individual
  → worst individual
  → $\sum f_i$ or $\bar{f_i}$

Algorithm of GA:

1. Initialize population
2. Get current fitness (+filtering)
3. Create offsprings ⟷ crossover
4. Mutations
5. "Survival of the fittest"
   ↳ update the population

DDE ⟺ Optimization

* Optimization landscape
* Evolutionary algorithms
* Genetic algorithms
* Genetic programing

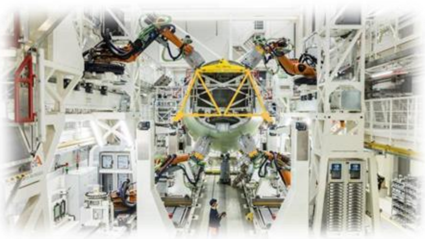
Data Driven Engineering

Obj: Engineering ⇒ Automate the process
↳ controllable way

DATA [source] → [mean] →

Craftsmanship
"handmade"

⇒



Automated production

Machine Learning
the same idea

Data Driven Engineering

Obj: Engineering ⇒ Automate the process
↳ controllable way

DATA [source] → [mean] →

Genetic Programming } Create a population of programs --Evolve--> Adapted programs

Experience the data on which they train.

Genetic Programming

* GA ⇒ fixed-length strings / lists / arrays

  - Stochastic decision process
  - Genetic operations { mutations, crossover, breeding }
  - Fitness based selection

* GP ⇒ hierarchical, variable in size

Gedanken Experiment

* EA ⇒ give a rod of certain length.

Rule : Rod must be assembled from smaller rods.

Ø $L_t = 9$ cm $l_1 = 1$, $l_2 = 2$, $l_3 = 4$ cm

Solution :

* $N < 3$
  $N > 9$

* eg. $N \to 5$;
  you get a subset of possible solutions.

Genome Size	# Sol.	Sample
2	Ø	–
3	3	4 4 1
4	12	4 2 2 1
5	20 5	4 2 1 1 1 2 2 2 2 1
6	6 120	4 1 1 1 1 1 2 2 2 1 1 1
⋮		
9	1	1 1
10	Ø	Ø

Data Driven Engineering

Genetic Programming

"Inductive learning"

GP → linear
→ graph_based
→ Tree representation

Year	Inventor	Technique	Individual
1958	Friedberg	learning machine	virtual assembler
1959	Samuel	mathematics	polynomial
1965	Fogel, Owens and Walsh	evolutionary programming	automaton
1965	Rechenberg, Schwefel	evolutionary strategies	real-numbered vector
1975	Holland	genetic algorithms	fixed-size bit string
1978	Holland and Reitmann	genetic classifier systems	rules
1980	Smith	early genetic programming	var-size bit string
1985	Cramer	early genetic programming	tree
1986	Hicklin	early genetic programming	LISP
1987	Fujiki and Dickinson	early genetic programming	LISP
1987	Dickmanns, Schmidhuber and Winklhofer	early genetic programming	assembler
1992	Koza	genetic programming	tree

Cramer, 1985 & Koza, 1989

* Suggested tree-like structure for program represent.
  " Genetic Programming "

In particular, I describe a single, unified, domain-independent approach to the problem of program induction – namely, genetic programming. I demonstrate, by example and analogy, that genetic programming is applicable and effective for a wide variety of problems from a surprising variety of fields. It would probably be impossible to solve most of these problems with any one existing paradigm for machine learning, artificial intelligence, self-improving systems, self-organizing systems, neural networks, or induction. Nonetheless, a single approach will be used here – regardless of whether the problem involves optimal control, planning, discovery of game-playing strategies, symbolic regression, automatic programming, or evolving emergent behavior.

J. KOZA, 1992

Why tree representation is popular?

* Recursive evaluation
* Dynamically changing sizes & shapes. (?)

! Allows algorithm to modify the structure of the solution

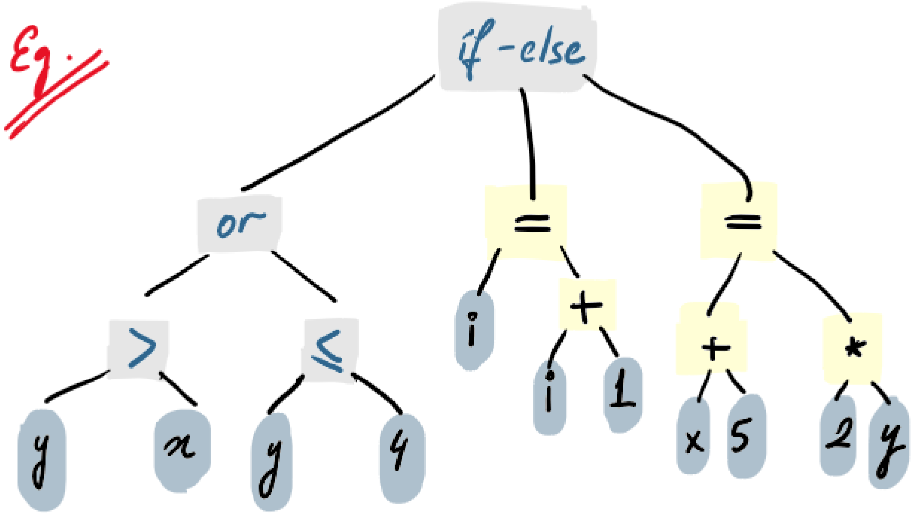
how many parameters?

Meaning of parameters

How parameters interact

Genetic Programming

Eg.



→ Evaluation ⇒ Recursive
[Depth-first >> left]

Karlsruhe Institute of Technology

Genetic Operations :

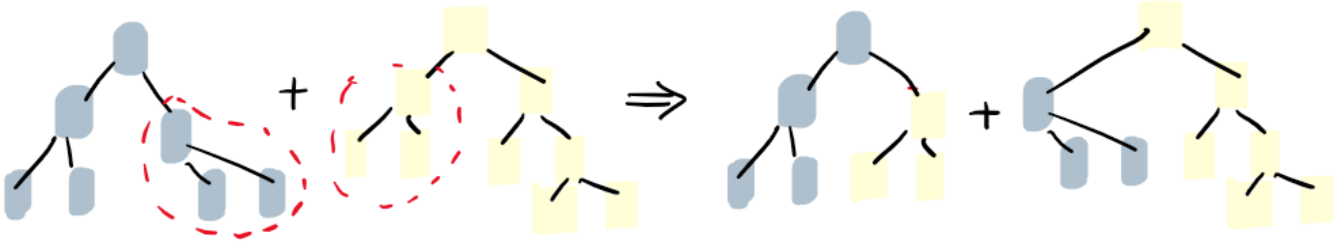
(1) Crossover

GA

Parent A Parent B

G.P

Single Point; Replace one subtree with the other.



Genetic Operations:

(2) Mutation

* Choose a node randomly.
  - ↳ Delete subtree
  - ↳ Replace it with a random tree.
  - ↳ Change a sub-property. ('>' ⇒ '<')

Genetic Programming

Karlsruhe Institute of Technology

* GA ⇒ fixed-length strings / lists / arrays

* GP ⇒ hierarchical, variable in size

Data structure   Algorithm

Functions   Terminals

Genetic Programming

Karlsruhe Institute of Technology

Terminal Set := Inputs to GP } at the end of every branch,

--> also includes constants (b_0)

Function Set := Statements Operators + Functions

- Boolean
- Arithmetic (+ − × /)
- Trancendental (Trigonometric, logarithmic)
- Variable assignments
- Conditionals (if..), controllers (go, jump..), loops (while..)
- Subroutines

Start basic & simple

How does structure evolve dynamically?

! "Iterative" + "Selective" algorithm

"Essence of evolution"

(i) population → reproduction opportunities

(ii) selection ⇒ better variants have higher chance

+

"Cumulative Selection"

? Practical significance in GP

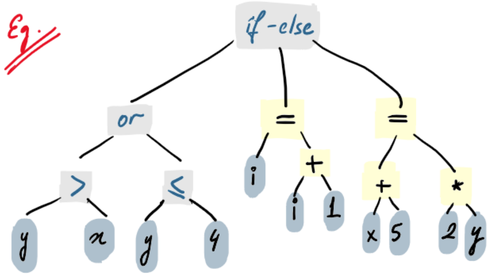
→ evolvable problem representation

→ Introns

Evolvable Representation

* Problem representation in ML ⇒ "constrained problem"
  [model is created & tailored]

Eg.



* Evaluation ⇒ Recursive
  [Depth-first >> left]

* GP ⇒ may evolve any solution (using Turing complete language)

Evolvable Representation

- GP may ignore operators / terminals

  e.g. $\{+, -, \times, /\}$; Fitness of "/" is typically ower.

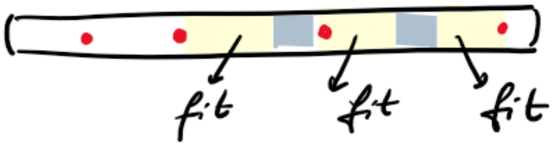
  $\rightarrow \{+, -, \times\}$

- Meta-learning ⇨ Create a bias for grammar rules from previous generations

- GP can find solutions of "right" length.

Introns ⇔ Bloat :

- Junk genetic materials $\left(\begin{array}{l} x = x * 1 \\ y = y + 0 \end{array} \right)$
  - ~90s; emerge due to variable length of GP genes.
  - GP ⇒ grows uncontrollably (until d_{max}).



"hitchhiking"

- less useful genes ⇒ spread into pool ⇒ reach the whole population } "Stagnation in evolution"

Introns ⇔ Bloat :

Introns ⇒ does not affect individual fitness.
Why do they emerge?

Effective fitness : Survivability of an indiviual's offspring

Introns ⇔ Bloat :

- Cross overs ~ children are much less fit than parents.
  Mutations ~ usually have (-) effects.
- Any parent reduces negative effects of (cross over / mutation) ↑
- Better a parent can protect the child from destructive operations ⇒ higher effective fitness

Introns ⇔ Bloat :

Introns ⇒ does not affect individual fitness.
Why do they emerge?

Fitness → Parent := chosen for reproduction
Fitness → Child := gene is passed down.

} Emergence of Introns

- Destructive genetic operators ⟹ create an advantage for parents with introns

Effective Complexity :

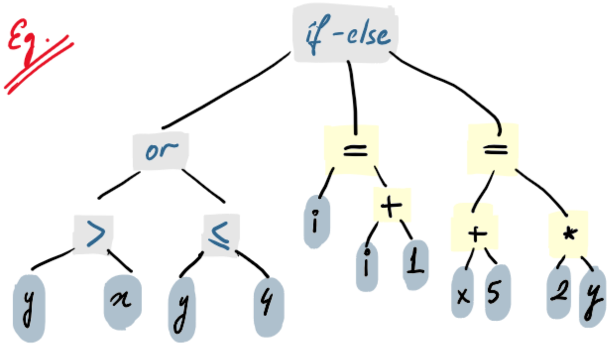
* complexity ⇒ # nodes ; size

$$\text{Eff. comp}(\mathcal{E}) = \text{useful genes} \,/\, \Sigma g_i$$

$$P_j^{t+1} = P_j^{t} \, \frac{f_j}{\bar{f}} \left(1 - p_c \cdot \mathcal{E} \, \rho_j^{d} \right)$$

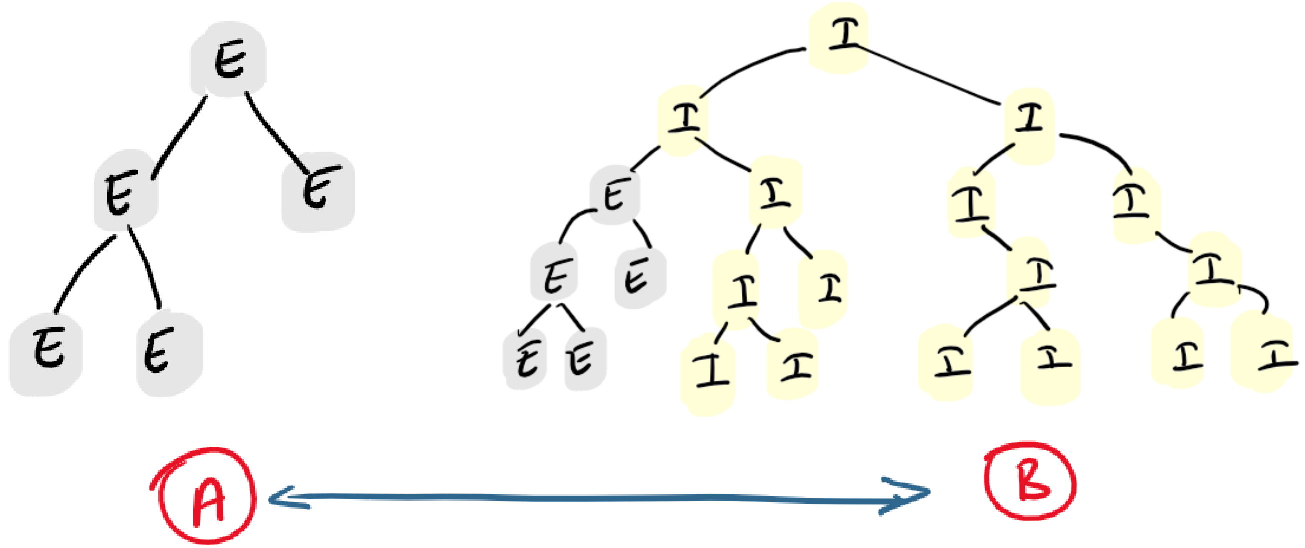
genes fitness crossover destructive

Eg.



* Evaluation ⇒ Recursive
[Depth-first >> left]

Effective Complexity :



$A^* \ll B^*$

Effective Complexity:

* As I % increases in population;

Destructive crossover ⟹ Neutral crossover

$\underbrace{\text{Destructive crossover} \Rightarrow \text{Neutral crossover}}$

Exchanged code has no/little effect.

- Strategy: Finding better solutions ⟹ Prevent disrupting good solutions

- Stagnation; more | comp. power / memory } effective growth ends.

What can be done?

- Reduce destructive effects of crossover → brood recomb. / → intelligent crossover ...
- Parsimony ⇒ penalty to the length of programs
- Variable fitness function → gradually
  - → "sensors"
  - → epochs

Genetic Programming

Convergence

* Max. generation
* Elapsed time
* Track fitness → Best individual
  → worst individual
  → $\sum f_i$ or $\bar{f_i}$

Algorithm of GA:

1. Initialize population
2. Get current fitness (+filtering)
3. Create offsprings ⟷ crossover
4. Mutations
5. "Survival of the fittest"
   ↳ update the population

GP: Initialization

* Heuristically // randomly
* max depth (d_{max} → hyperparameter)

↓

Full method		Grow method	→ ramped half-half
• $d < d_{max}$ node → function (f)	)	• $d < d_{max}$ node → f // t	
• $d = d_{max}$ node → terminal (t)		• $d = d_{max}$ node → t	

Closer look to Crossover :

* Primary search mechanism for opt. problem

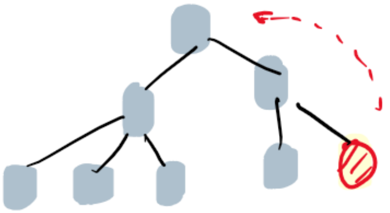
Hypothesis ⇒ good building blocks combined into larger, better blocks. ?

☐ Algorithm ⇒ Take the fittest parents

⇒ Combined ⇒ better individuals !

Closer look to Crossover :

! Good building blocks ⇒ dominant ⇒ reflected on fitness.



Can we generalize it ?

* Crossover ≈ 70–75 % lethal to offsprings

* Nature

⇒ restricting mating of intraspecies

~ viable offsprings

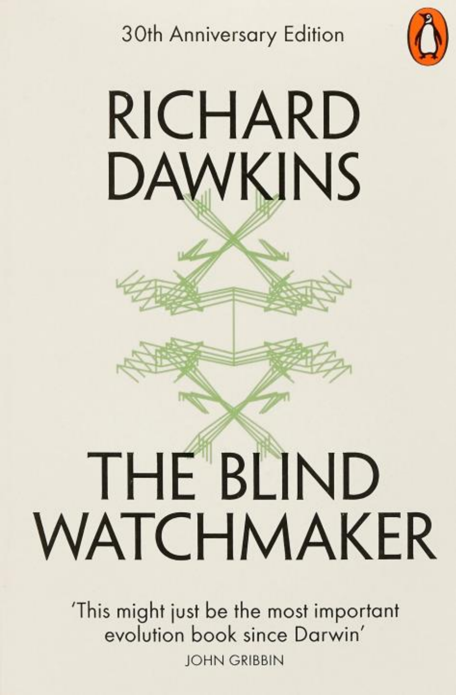
⇒ "Semantics"

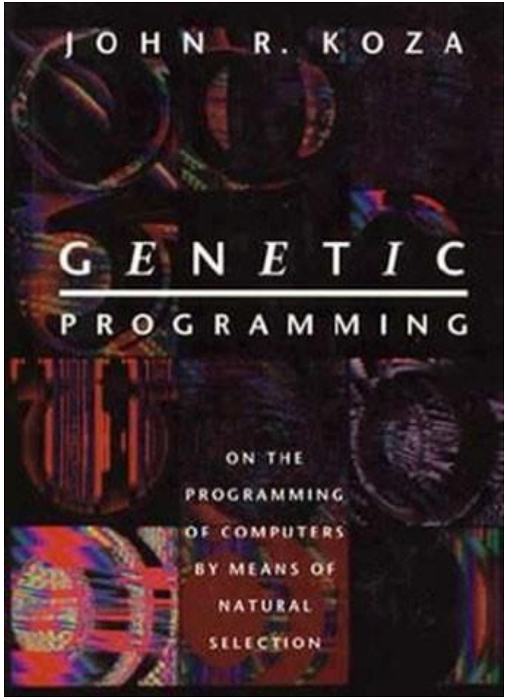
hair color ⇔ muscle density

⇒ "homologous"

~ very similar down to molecular level

colab





- Handbook of Genetic Programming Applications, Springer
- Genetic Algorithms and Genetic Programming Modern Concepts and Practical Applications, CRC