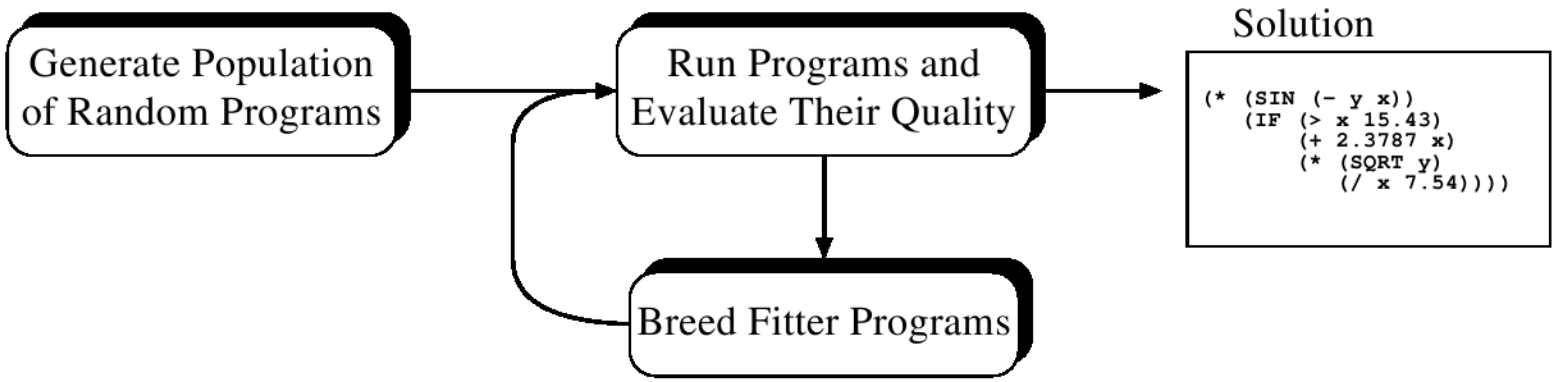
Genetic Programming

after *A Field Guide to Genetic Programming* by Riccardo Poli, and more about NEAT and MOSES

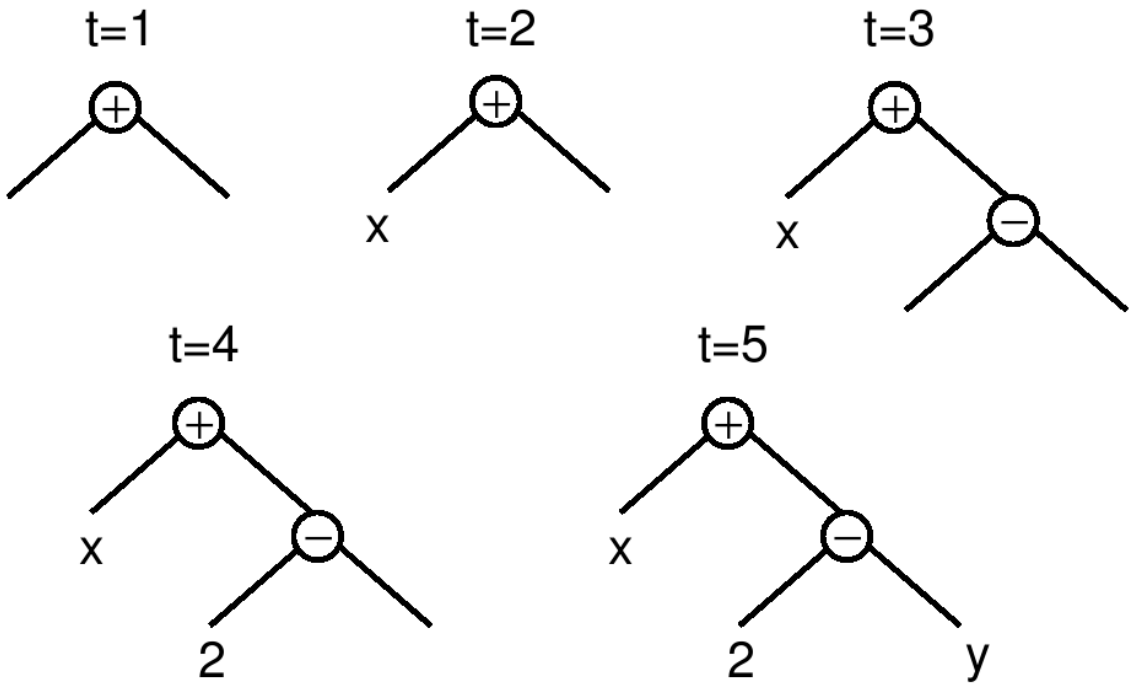
BY ŁUKASZ STAFINIAK

Introduction to Genetic Programming

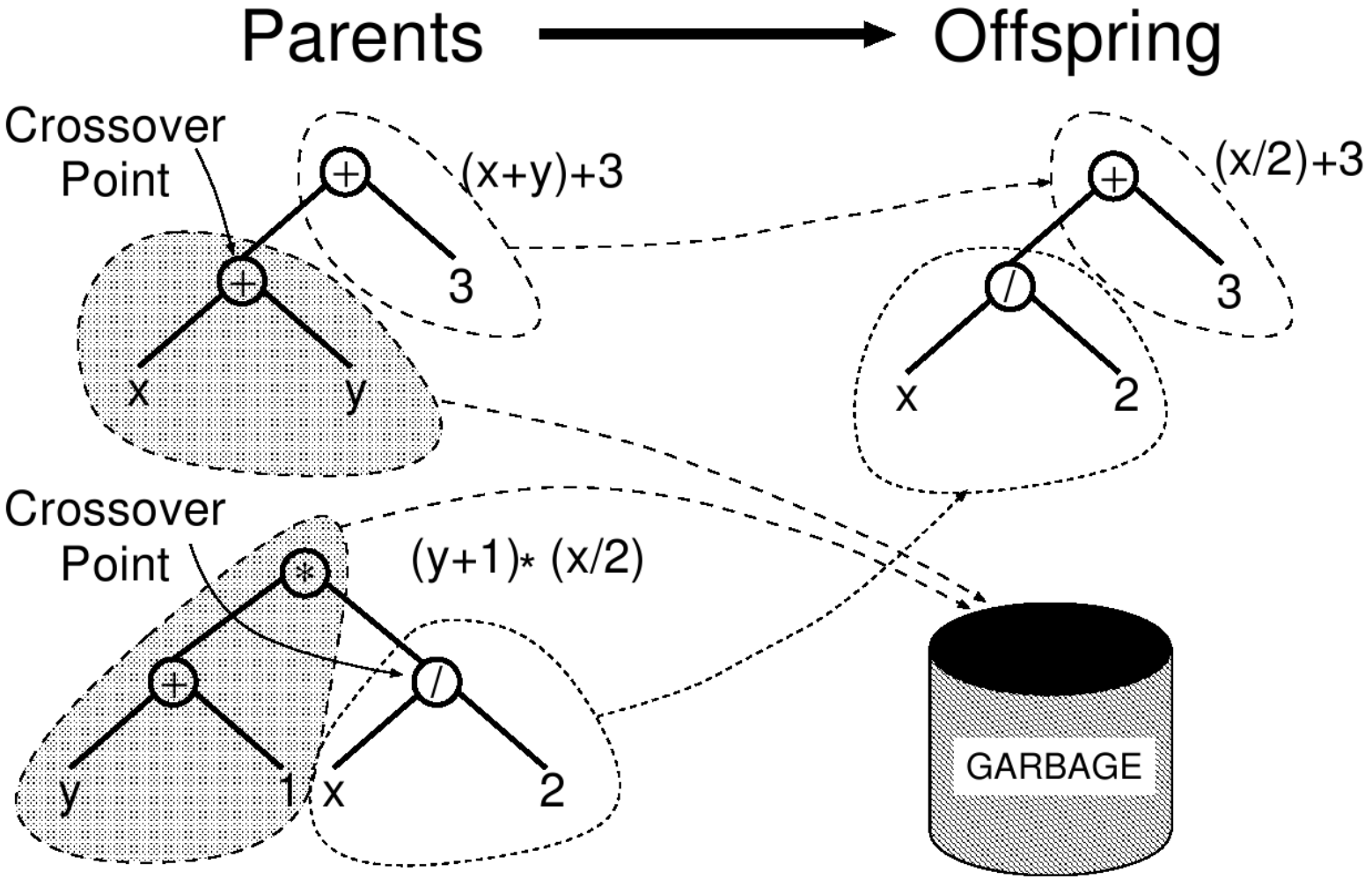


Basic Program Generation

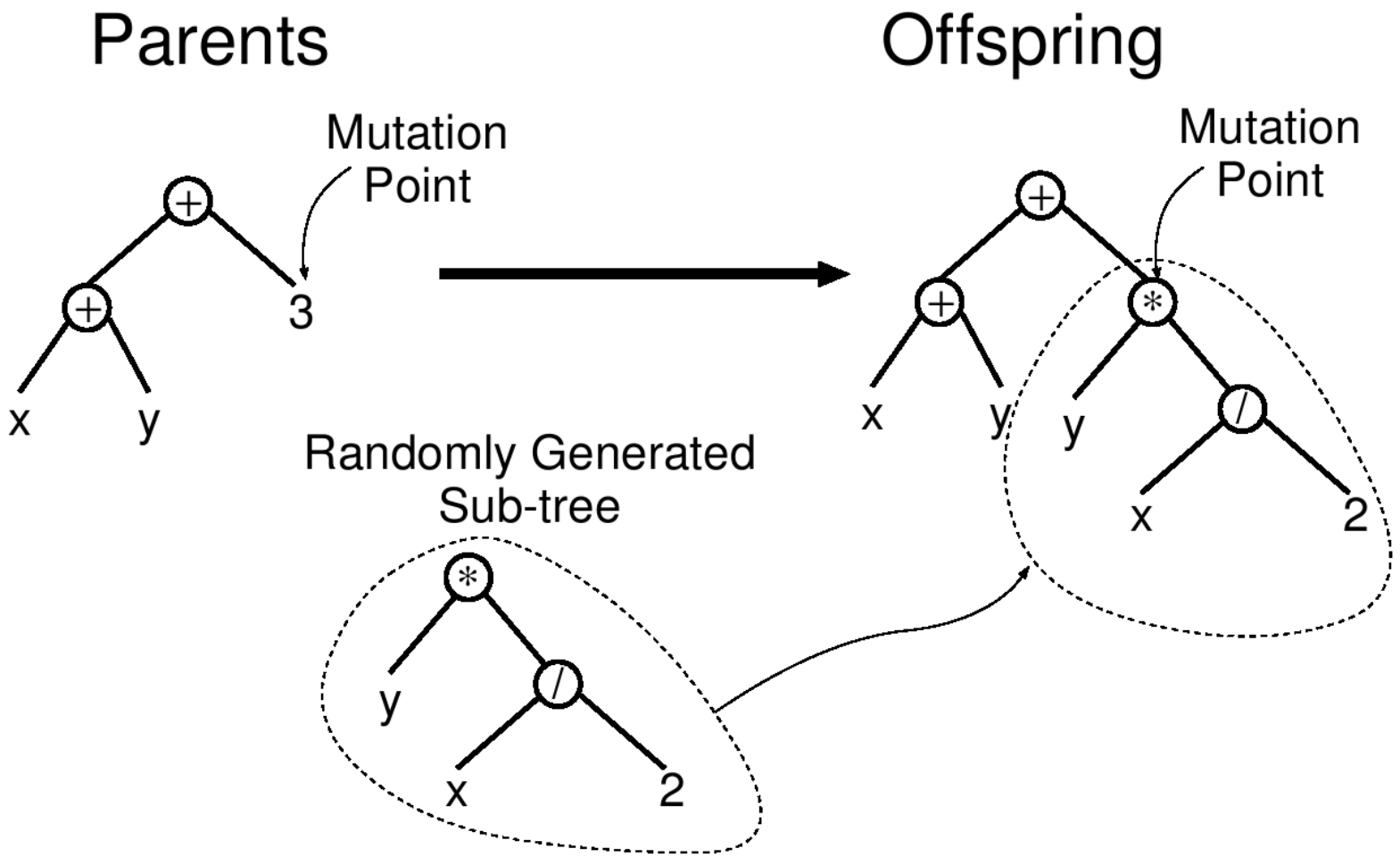
- Fixed depth • Maximal depth
- Maximal length (a bit more difficult)
- Ramped half-and-half: half population – fixed depth, half – maximal depth, different depths used



Recombination



Mutation

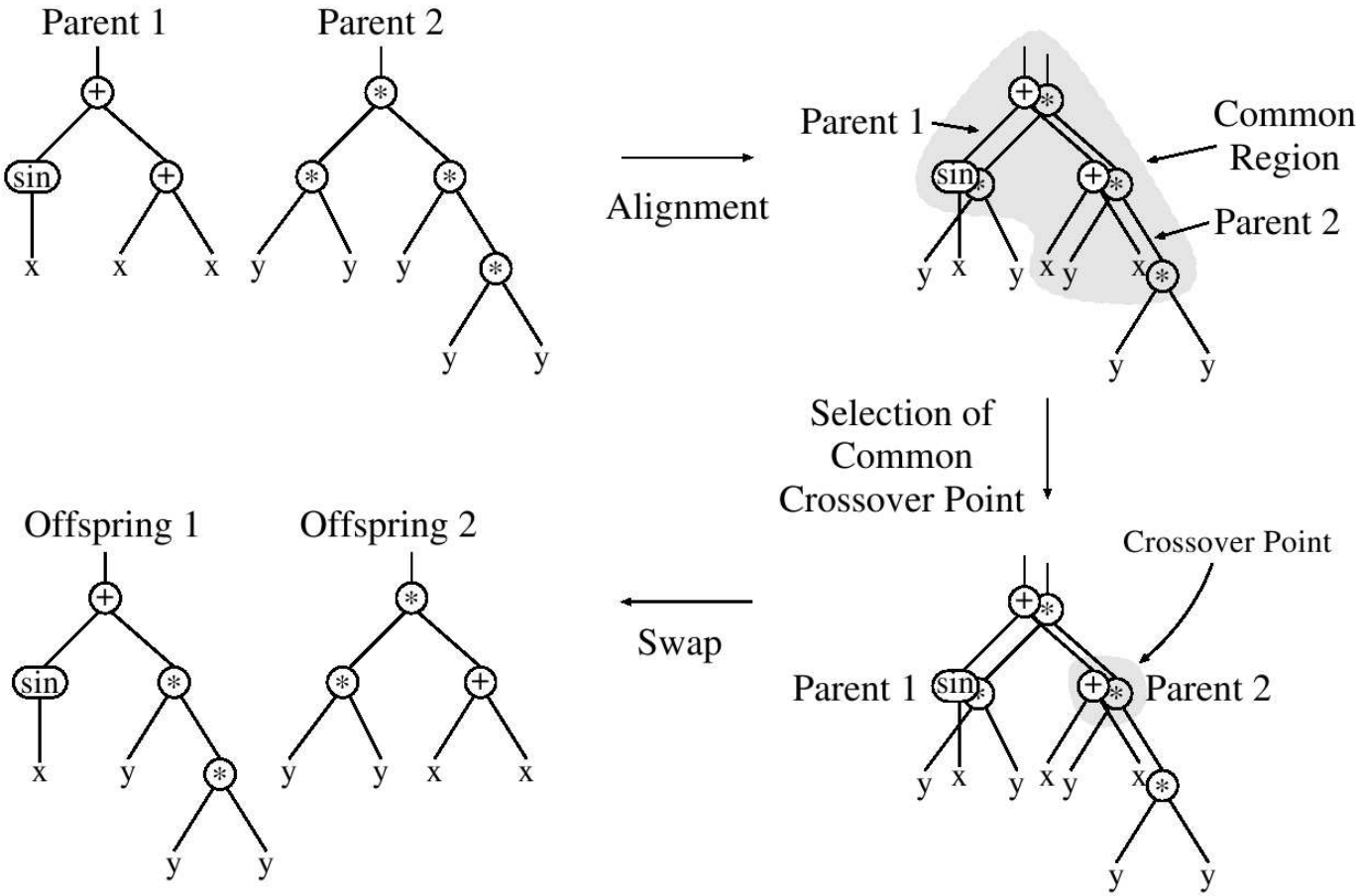

Parents
Offspring
Mutation
Point
Mutation
Point
3
x
y
x
y
y
x
2
Randomly Generated
Sub-tree
y
x
2

Alternative Initialisations and Operators in Tree-based GP

- Uniform initialisation produces more assymetric trees than ramped half-and-half.
- Crossover tends to create short programs (so called Lagrange distribution); short programs surprisingly faciliate code bloat.
- Mutation kinds:
  - **size-fair subtree mutation**
  - **node-replacement mutation**
  - **hoist mutation** promotes a subtree as the result
  - **permutation mutation** swaps arguments of a node
  - **mutating constants systematically** by numerical optimization

Crossover

- one-point
- uniform
- context-preserving
- size-fair
- crossover ops specific to other representations



Modular and Grammatical Tree-based GP, Linear GP

- Modularity can be introduced either by extracting a global library of subroutines, or locally by evolving program-local functions.

- Automatically Defined Functions are attached to individuals, recombination works within corresponding ADFs or Result-Producing Branches (e.g. ADF2 from A crossed-over with ADF2 from B). To forbid looping, e.g. ADFi can call ADFj for $i < j$.

    - Also a. d. iterations (ADIs), a. d. loops (ADLs) and a. d. recursions (ADRs) provide means to reuse code. A. d. stores (ADSs) provide means to reuse the result of executing code.

- Architecture-altering Operators can introduce and remove ADFs.

- **Strongly-typed GP**.
    - Similar trade-offs as with constrained optimization in GAs.

- Grammatical Evolution uses variable-length integer sequences that encode rule choices when generating a phenotype from the user-supplied grammar.
    - Rules for a nonterminal selected "modulo", ignore unused tail of a sequence, wrap or fail when sequence too short.

- Linear GP uses variable-length sequences of instructions as individuals.

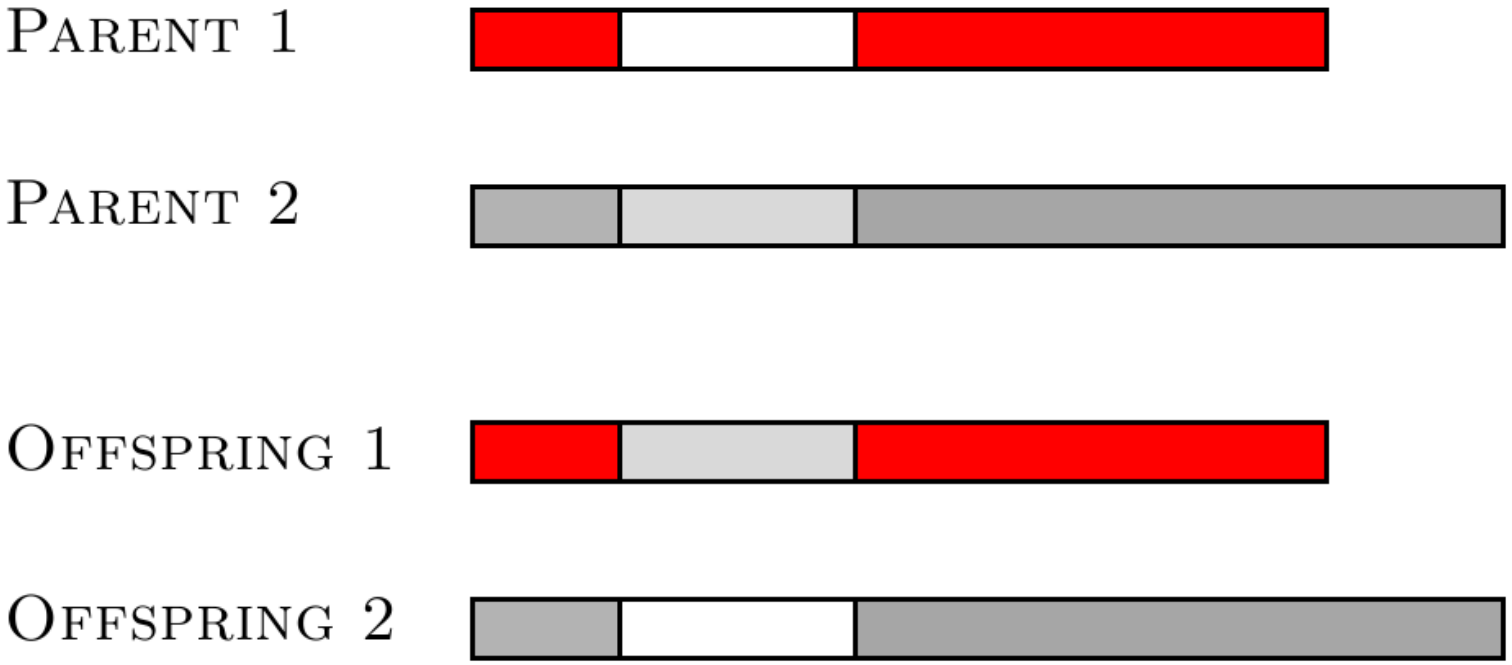
Two-point crossover affects length.

Parent 1

Parent 2

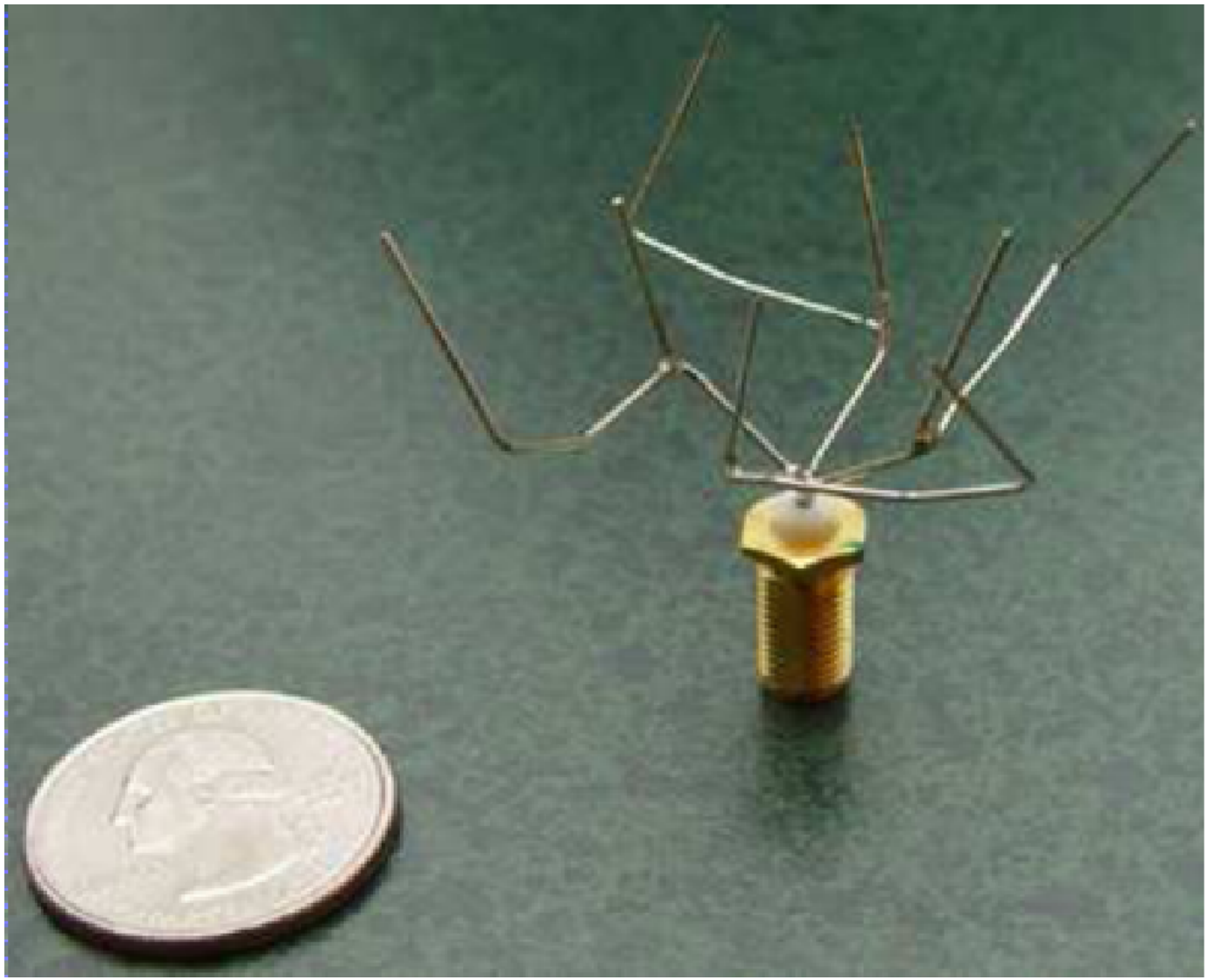
Offspring

"Homologous" crossover preserves lengths.



Applications

Human-competitive antenna design produced by GP:



Some Theory

Code Bloat

Some explanations of code bloat:

- replication accuracy theory: bigger size lets children be more functionally similar to parents
- removal bias theory: inactive subtrees of fit programs are small and close to leaves, replacing them with random size subtrees preserves fitness and grows size
- nature of program search spaces theory: there is more bigger programs with the same fitness as smaller ones – sampling effect
- crossover bias theory: while crossover keeps the expected size of programs constant, it pushes the distribution out of middle-ground, generating either shorter or longer programs, and short are unlikely to stay as fit

Neuro-Evolution

Neuro-evolution means creating Neural Networks by evolutionary algorithms.
Different apporaches:

1. Fix NN topology and use a GA or an ES to evolve weights
2. Evolve the NN as a graph: develop a subgraph-swapping crossover
3. Indirect encoding (artificial morphology):
    a. DeGaris' Cellular Automata
    b. Gruau's Cellular Encoding

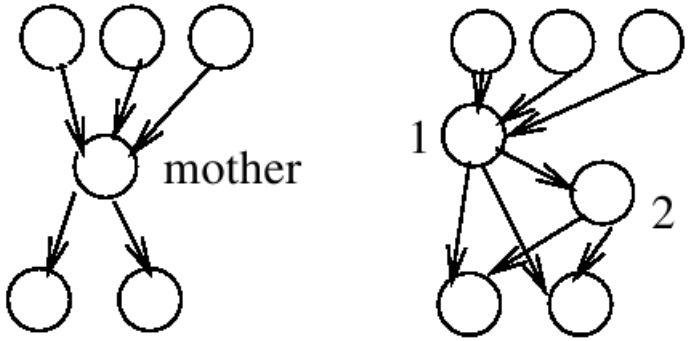
Framsticks

Framsticks uses several encodings, with neurons attached to body parts: "brain topology" refines the "body topology". Framsticks `f0` low-level encoding uses phenotype representation for crossover (cutting plane) (mixed encoding); `f1` encoding uses subtree (or substring) swapping crossover (direct encoding). `f4` uses indirect encoding based on Cellular Encoding.

Gruau's Cellular Encoding

- operations: cell (proto-neuron) divisions `PAR`, `SEQ`, `CPO`, and cell register modifications

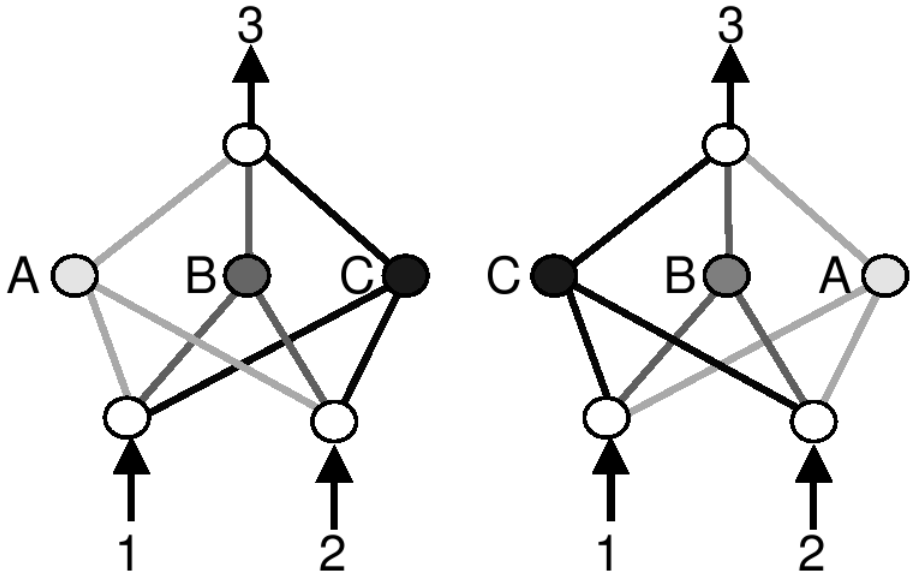
`CPO`:



- cells develop concurrently one operation at a time by a FIFO queue manager by executing a program tree
- at a `PROGN` node of the tree a cell executes all subtrees in sequence, at other nodes, after executing the node operation, it chooses a subtree
- crossover exchanges program subtrees

NeuroEvolution of Augmenting Topologies

Problem: Competing Conventions



[A,B,C]
X[C,B,A]

Crossovers: [A,B,A] [C,B,C]
(both are missing information)

Nature's solution utilizes homology: alleles of the same trait are homologous.
(E.g. in E. coli, a special protein RecA lines up homologous genes.)
Actual homology between NNs is difficult to compute.
The main insight of NEAT: find homology based on historical markings.

Protecting Innovation with Speciation.
Structural innovations need to be tuned before they are competitive.
In NEAT, distance between genomes is easy to compute: use speciation.
Explicit fitness sharing:

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} \mathbb{1}_{\delta(i,j) \leqslant \delta_t}}$$

Distance:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W}$$

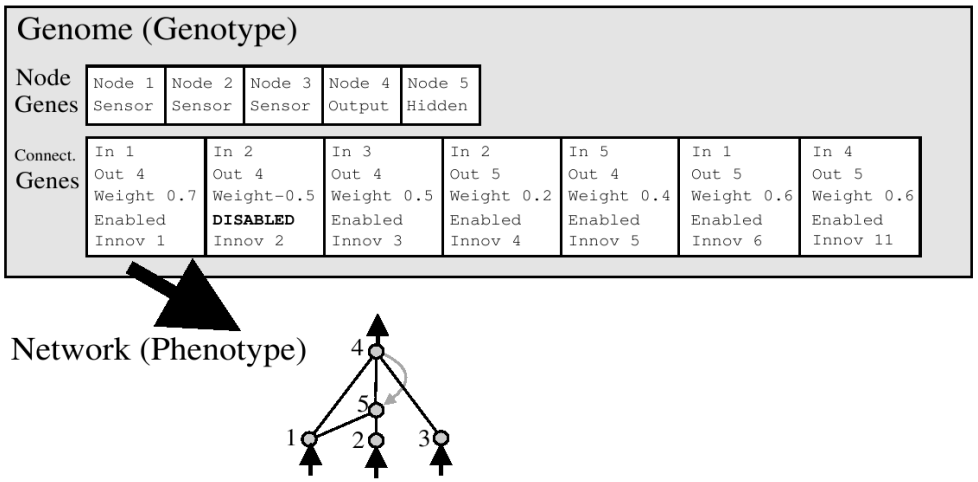
where

- N – the number of genes in larger genome,
- E – number of excess genes (intuition: they came after the lineage split),
- D – number of disjoint genes,
- $\bar{W}$ – distance of weights for matching genes.

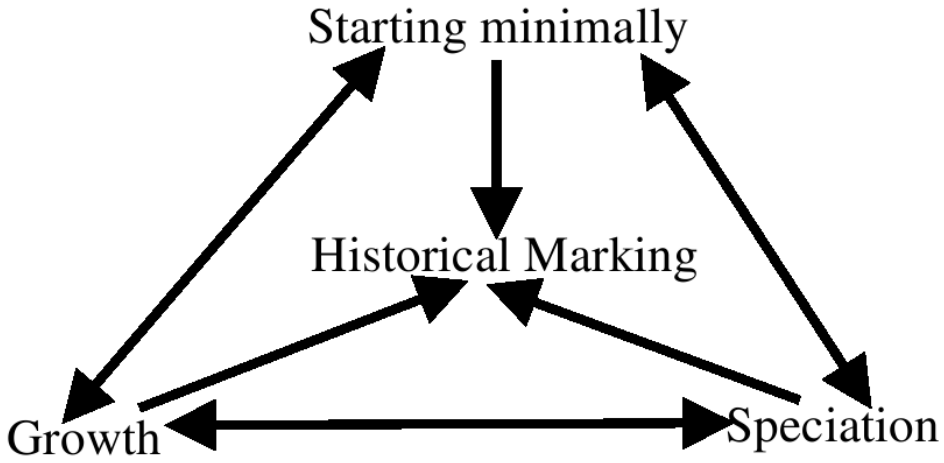
Protection of innovation allows NEAT to start from minimal structure: no hidden neurons. Only useful topology extensions are kept: this avoids bloat.

Historical (genealogical) markings and structure growth.
Structural neighborhood is processed effectively: a connection between fixed nodes is only introduced once.

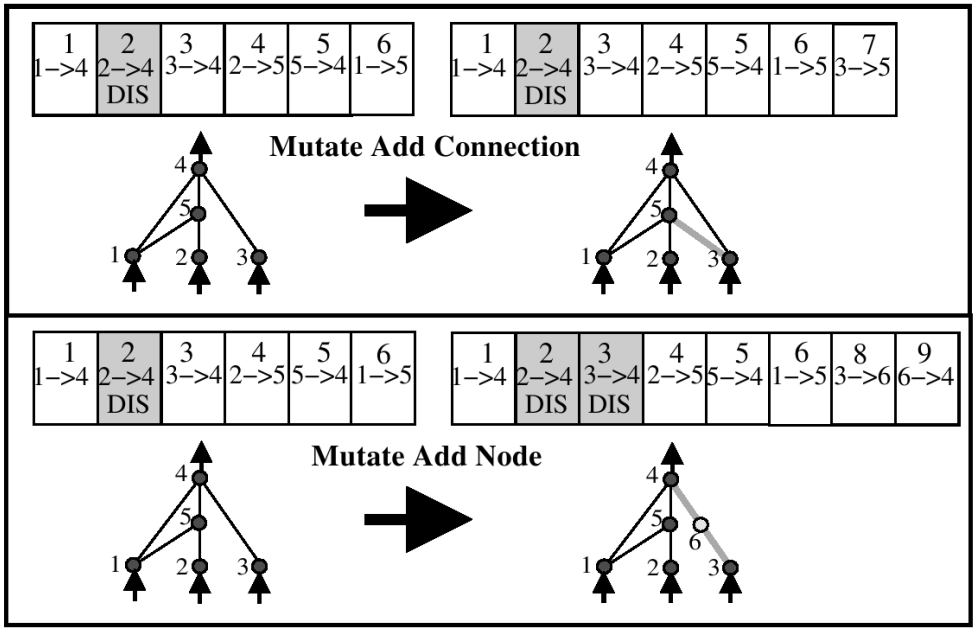
Representation:



NEAT "architecture":



Mutation:


1 1->4
2 2->4 DIS
3 3->4
4 2->5
5 5->4
6 1->5
1 1->4
2 2->4 DIS
3 3->4
4 2->5
5 5->4
6 1->5
7 3->5
Mutate Add Connection
1 1->4
2 2->4 DIS
3 3->4
4 2->5
5 5->4
6 1->5
1 1->4
2 2->4 DIS
3 3->4 DIS
4 2->5
5 5->4
6 1->5
8 3->6
9 6->4
Mutate Add Node

Crossover:

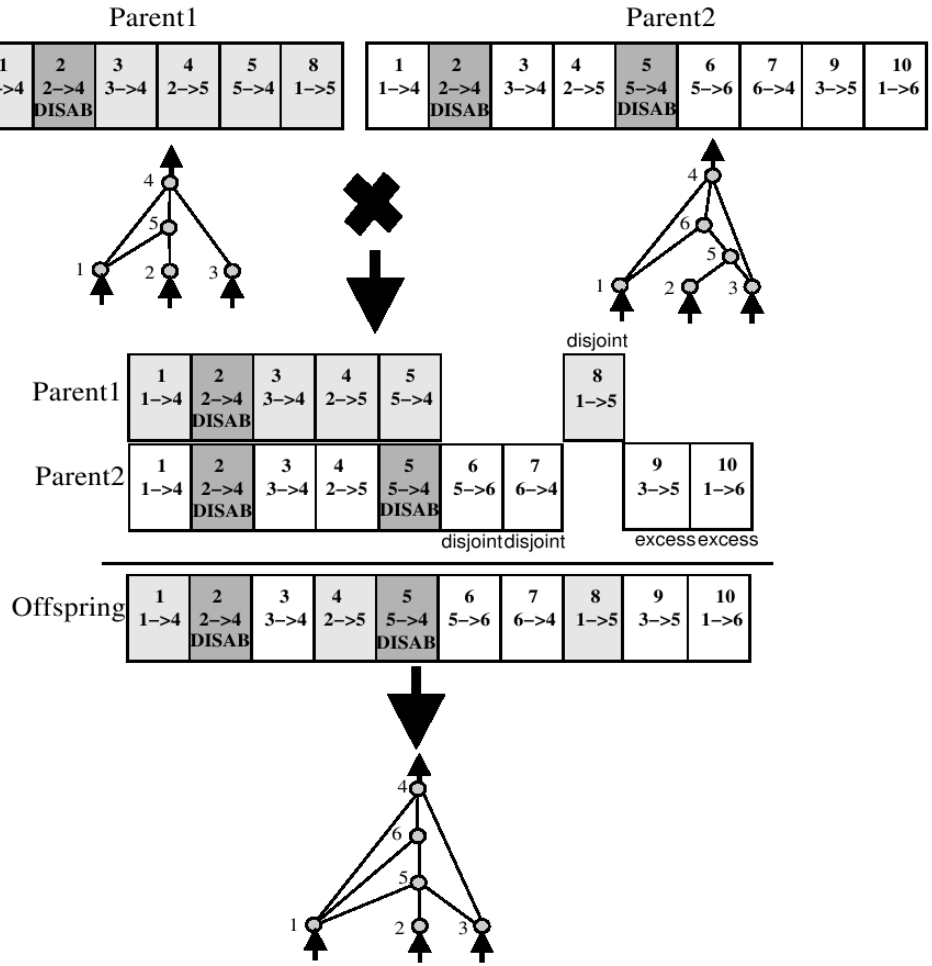
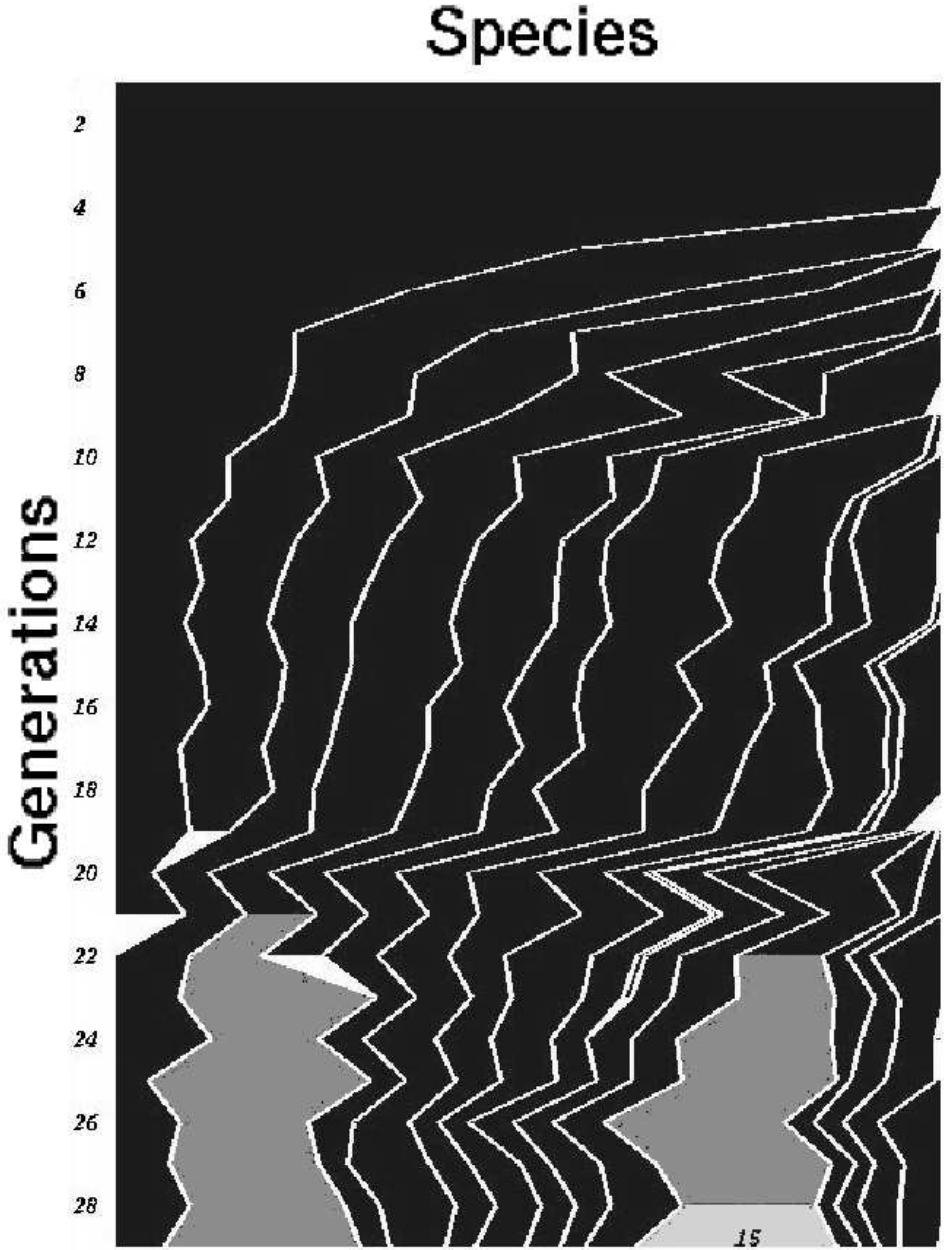

Parent1
Parent2
1 1->4
2 2->4 DISAB
3 3->4
4 2->5
5 5->4
5 5->4 DISAB
6 5->6
7 6->4
8 1->5
9 3->5
10 1->6
disjoint
disjoint
disjoint
excess
excess
Offspring

Illustration of evolution:



pole balancing problem

Meta-Optimizing Semantic Evolutionary Search (MOSES)

- explores the space of arbitrary programs but exploits domain-specific knowledge, which can be combined from more-or-less generally applicable libraries
- has the "Starting minimally-Growth-Speciation" traits of NEAT on a whole new level
- is organized in demes, each deme is a combinatorial space built around an exemplar program from representation-building domain-specific generators which are modifications minimal both syntactically and behaviorally (semantically)
    - e.g. $x \Longrightarrow 0 + x$ or $x \Longrightarrow 1 * x$, which can later evolve into behaviorally very different $-x$

Normalization

- exemplar programs are kept in domain-specific normal form, which also governs representation-building generators
    - e.g. Elegant Normal Form for boolean formulas which preserves hierarchical structure (over OR, AND, NOT)
        - negation only in literals, conjunction and disjunction alternate
        - no node has multiple occurrences of the same variable's literal as children
        - no two literals of a variable are conjuncted (x and x or $\sim x$ have $\vee$ as youngest common ancestor)
        - common literals in a node are factored out
        - no set of literal children of a node is a subset of a sibling's set
        - no third sibling's set (of literal children) is a subset of $S_1 \cup S_2$ if two siblings' sets are $\{x\} \cup S_1$ and $\{\sim x\} \cup S_2$
- normalization correlates syntactic and semantic distances

- each possible knob (generator application) is checked for redundancy (if program reduces size by normalization)

More

- within-deme evolution by hBOA
- to attack harder problems, additional subsampling of $\sqrt{n}$ programs at distances up to $\sqrt{n}$ from the exemplar
- approximate bounds for time complexities (one deme):
  - for a deme $O(N\,n^3)$, where N is the population size, $n = O(a\;l)$ is the size of representation (a is the arity of the function set, l is the size of the exemplar)
  - from which n^2 for representation building and n for number of generations
  - letting $N = O(2^k n^{1.05})$, where k is the highest degree of subproblem interaction, and $c(l)$ the cost of scoring a function

$$O(N(n\,c(l) + n^3)) = O(2^k\,(a\,l)^{2.05}\,(c(l) + (a\,l)^2))$$