

Ontogenetic Programming

Lee Spector*†
Kilian Stoffel †

*School of Cognitive Science and Cultural Studies, Hampshire College, Amherst, MA 01002

†Department of Computer Science, University of Maryland, College Park, MD 20742

This talk also includes results reported in Spector, L., and K. Stoffel. 1996. Automatic Generation of Adaptive Programs. In P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S.W. Wilson (editors), *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. Cambridge, MA: The MIT Press.

Overview

- Phylogeny and Ontogeny
- Ontogenetic HiGP
- Examples:
 - Binary Sequence Prediction
 - Wumpus World
- Conclusions

Phylogeny and Ontogeny

- *Phylogeny* = the developmental progression of a population through evolutionary time.
- *Ontogeny* = the developmental progression of an individual throughout its lifespan.
- GP uses biologically inspired phylogenetic mechanisms.
- Through the addition of ontogenetic mechanisms, GP can produce adaptive programs that solve more difficult problems.

Ontogeny and Morphology

- *Morphology* = the developmental progression of an individual from genotype to phenotype. (“growth phase”)
- Morphological components in GP include Gruau’s encoding → network transforms, Zomorodian’s tree → PDA transforms, and Spector’s ADM expansions. See [Angeline 1995] for formal definitions and a survey.

Ontogeny and Morphology

- *Ontogeny* = the developmental progression of an individual *throughout its lifespan*. **Note** that this development may be guided by the runtime environment.
- Morphology \subset Ontogeny.

Ontogenetic Mechanisms

- Runtime memory mechanisms:
 - Indexed memory [Teller 1994]
 - Memory terminals [Iba et al. 1995]
 - Runtime “morphology” implemented via program self-modification operators. We call this strategy ontogenetic programming.

HiGP

[Stoffel and Spector 1996]

- A high-performance GP system .
- Manipulates linear (rpn) programs that are executed on a stack-based virtual machine.
- Fast, flexible, and portable.
- Parallel HiGP scales nearly linearly with the number of available processors.
- Program self-modification mechanisms can be particularly simple for linear programs.

HiGP

Virtual Stack Machine Example

```
push-x noop push-y * push-x push-z  
noop - + noop noop
```

The noops in this program have no effect and the remainder is equivalent to the Lisp expression:

```
(+ (* x y) (- x z))
```

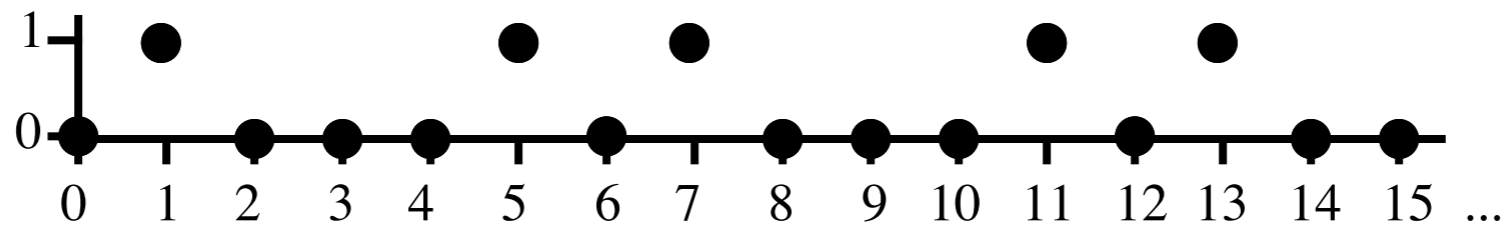
and to the C expression:

```
(x * y) + (x - z)
```


Ontogenetic HiGP

- **segment-copy** copies a part of the linear program over another part of the program. The function takes 3 arguments from the stack: the start position of the segment to copy, the length of the segment, and the position to which it should be copied.
- **shift-left** rotates the program to the left. The call takes one argument from the stack: the distance by which the program is to be rotated.
- **shift-right** rotates the program to the right. The call takes one argument from the stack: the distance by which the program is to be rotated.

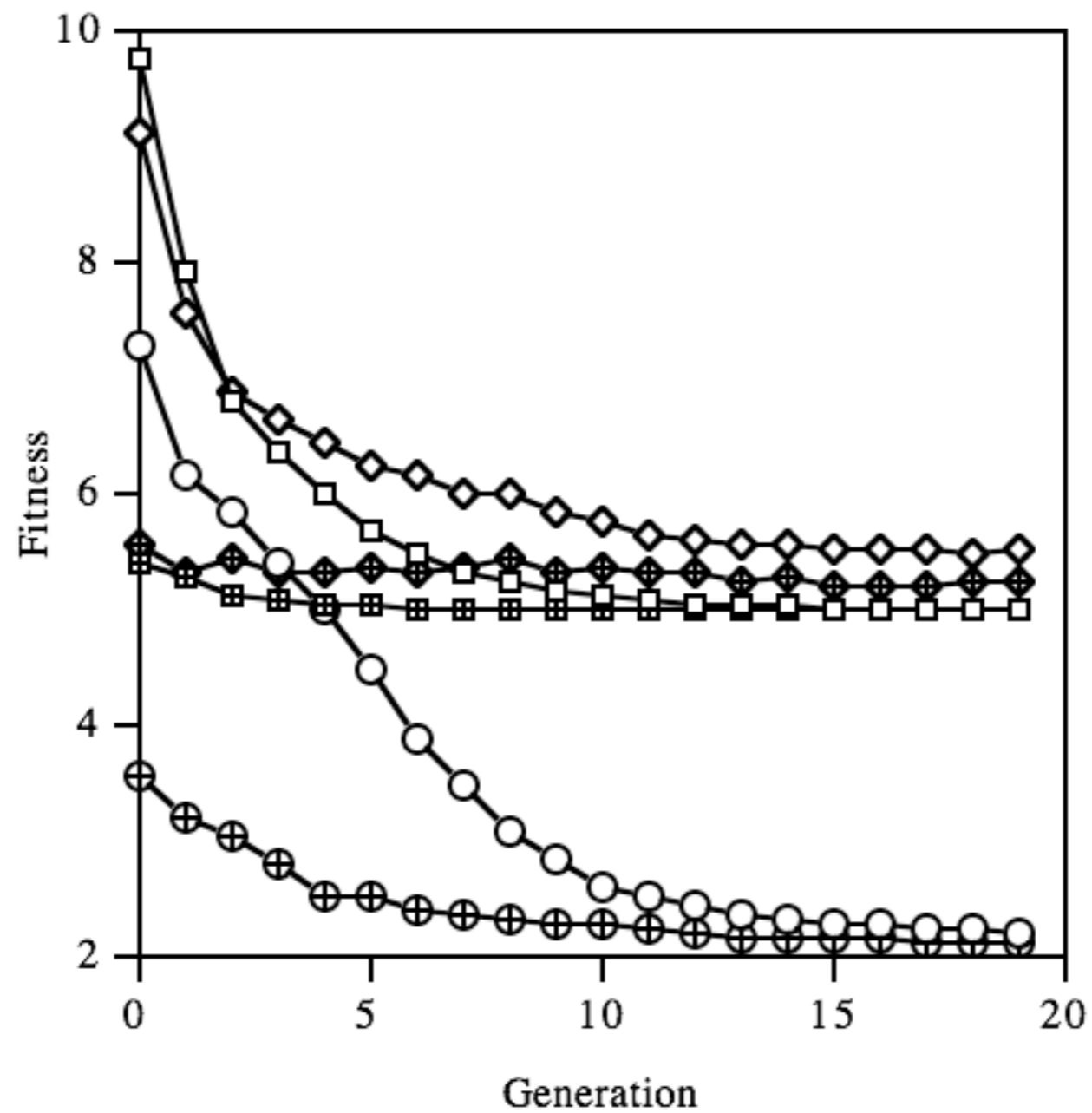
Binary Sequence Prediction



- As in symbolic regression, attempt to evolve a function of x that produces the corresponding y .
- Run programs on a sequence of x values (0–17 here), always in the same order, for each fitness test “lifetime.”

Binary Sequence Prediction

- Function set: +, -, *, ÷, push-x, [ontogenetic operators], [memory operators]
- Positive output \Rightarrow 1, 0 or negative \Rightarrow 0
- Population size=100, Program size=30, 90% Crossover, 10% Reproduction, 20 Generations



- Non-ontogenetic Average Average
- Non-ontogenetic Average Best
- Ontogenetic Average Average
- ⊕— Ontogenetic Average Best
- ◇— Indexed Memory Average Average
- ◆— Indexed Memory Average Best

Results

- In 100 runs *without* ontogenetic operators, no successful programs were evolved.
- In 100 runs *with* ontogenetic operators 12 successful programs were evolved. Of these 10 appeared to be general; although fitness was assessed only over the range [0–17], these programs produced correct results over the range [0–39].
- In 100 runs with indexed memory (and without ontogenetic operators) *no* successful programs were evolved.

Snapshots Over a Lifetime

```
+ push-x - noop - shift-left shift-right push-x  
segment-copy shift-left shift-right * + segment-copy +  
% * * noop noop shift-right + % shift-left shift-right  
noop noop - noop *
```










After the completion of 9 full executions:

```
- shift-left + % * * noop noop shift-right % shift-left  
noop noop - noop * + push-x - noop - shift-left + % * *  
noop noop shift-right %
```

At the end of 18 executions it appears more similar to, but still different from, its initial state:

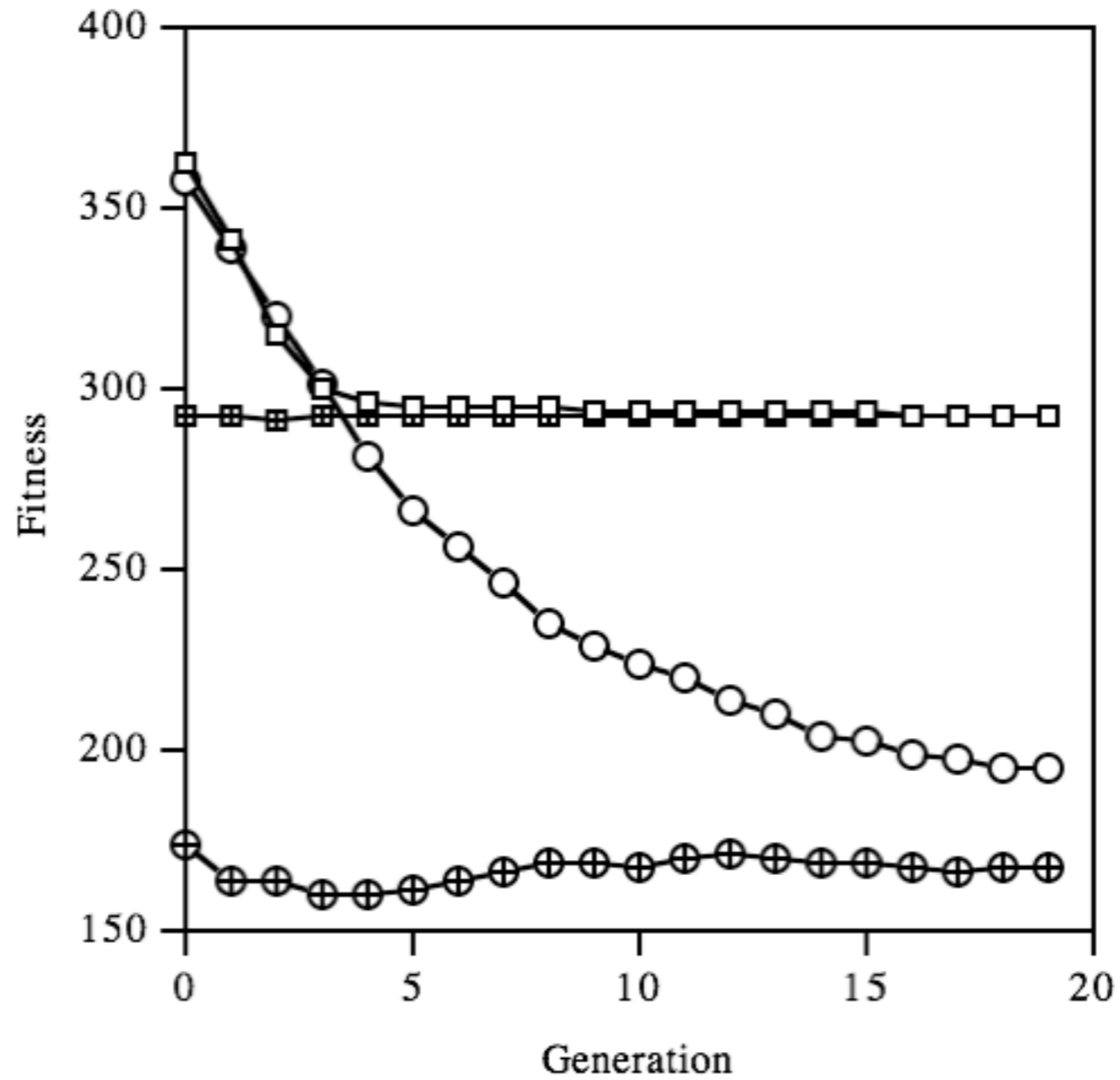
```
+ push-x - noop - shift-left shift-right segment-copy  
shift-left * + segment-copy + % * * noop noop shift-  
right % shift-left noop noop - noop * + push-x - noop
```

Wumpus World

Breeze	 Pit	Breeze		Breeze	 Pit
 Pit	Breeze			Breeze	 Pit
Breeze		Breeze			Breeze
	Breeze	 Pit	Breeze Stench		 Gold
		Breeze Stench	 Wumpus	Stench	Breeze
 Agent			Stench	Breeze	 Pit

Wumpus World

- Goal: to guide an agent through a complex and dangerous virtual world (Russell and Norvig, 1995).
- Function set: and, or, not, sequence, if-zero, if-less-or-equal, +, -, *, sensors, constants, [read, write]
- Population size=200, Program size=100, 89.5% crossover, 10% reproduction, 0.5% mutation, 20 generations per run



- Non-ontogenetic Average Average
- Non-ontogenetic Average Best
- Ontogenetic Average Average
- ⊕— Ontogenetic Average Best

Results

- In 200 runs *without* ontogenetic operators, *no* successful programs were evolved.
- In 200 runs *with* ontogenetic operators 10 successful programs were evolved.

Evolved Program

```
noop and 3 write - - not + and + 3
2 noop 6 * write 5 4 1 ifz 1 + + 6
read 1 and + + 2 shift-right 1
stench breeze + or * 0 breeze + or
1 2 4 shift-left 3 bump not 1 ifz 0
1 6 read glitter 5 segment-copy not
3 shift-left shift-right write
write * stench - 6 bump sound - 6
noop bump glitter 0 3 - bump 0 0
sound bump stench 4 * or 1 ifz and
5 2 bump 5 * 5 write 6 and 1 -
```

Ontogenetic Programming with S-Expressions

- subtree-copy (from-index, to-index)
 - between rather than during executions
 - global indices not meaningful after crossover
 - explosive ontogenetic growth
- structured-subtree copy (from-index, to-index, rpb)
- dynamic ADFs and ADMs
 - versions of defun, funcall etc. in function set
 - store functions/macros in indexed memory
 - runtime self-modification via module redefinition

Future Work

- Real-world problems
- What self-modification strategies are actually used by successful individuals?
- Vary set of ontogenetic operators

Conclusions

- GP can be used to produce programs that themselves develop in significant, structural ways over the course of a run.
- “Ontogenetic programming” is the technique of including program self-modification operators in the function set.
- Ontogenetic programming can allow for the evolution of solutions in cases for which ordinary GP fails.