Multi-type, Self-Adaptive Genetic Programming as an Agent Creation Tool

Lee Spector, Hampshire College
Alan Robinson, Hampshire College/UCSD
lspector@hampshire.edu, http://hampshire.edu/lspector

Thanks also to Jon Klein.
Overview

Goals

Technologies: Push, PushGP, Pushpop, Breve (by Jon Klein)

Results:

- **PushGP**: Evolved transport network agents
  Evolved “Opera” agents
  Confirmed/extended Van Belle/Ackley effect

- **Pushpop**: Reliable auto-diversification

- **Breve**: Evolved goal-directed 3D swarms [DEMO]

Future
1. Provide technologies for the automated production of agents for complex, dynamic environments.

2. Develop self-adaptive (self-configuring) evolutionary computation systems in the service of Goal #1.

3. Investigate general properties of self-adaptive evolutionary systems using the technologies developed for Goal #2.
The Push Programming Language for Evolutionary Computation

Designed for the expression of evolving programs within an evolutionary computation system.

Simplifies the evolution of agents that may use:
• multiple data types
• subroutines (any architecture)
• recursion
• evolved control structures
• evolved evolutionary mechanisms

Push supports all of this using simple, mostly standard GP techniques.

Stack-based language with one stack per type; types include integer, float, Boolean, code, child, type, name.
Push

Stack-based, like Forth or Postscript

Multiple stacks, one for each type

Types are hierarchical

**Type** constants on a **type** stack/bottom

Missing argument? NOOP

**Code** type/stack -> advanced features

Runtime resource limits
Push Architecture

possibly nested program of stack-manipulating instructions

integer stack  float stack  Boolean stack  code stack  type stack  URL stack  name stack

more stacks as needed
Push Examples

(integer 2 3 +)

(integer 2 3 + float 2.72 3.14 +)

(2 3 2.72 3.14 integer + float +)

(2.72 integer 2 3.14 3 + float +)

((integer) (2 (3)) +)

(code quote (integer 2 3 +) do)
Factorial in Push

(quote (pop 1)
quote (code dup
    integer dup
    1 - do *)
integer dup 2 < if)
(code
  quote (quote (pop 1)
    quote (integer dup 1 -
      code factorial get do
    *)
    integer dup 2 < if)
  factorial set
  factorial get do)
The Push Type Hierarchy

- push-base-type: dup, pop, swap, rep, = [boolean],
  set[name], get[name], convert[type],
  pull[integer], noop
- number: +, -, *, /, > [boolean], < [boolean]
  - integer: rand, pull, /
  - float: rand
- boolean: not, and, or, nand, nor, rand
- expression: quote, car, cdr, cons, list, append, subst,
  container, length[integer], size[integer],
  atom[boolean], null[boolean], nth[integer],
  nthcdr[integer], member[boolean],
  position[integer], contains[boolean],
  insert[integer], extract[integer],
  instructions[type], perturb[integer],
  other[integer], other-tag[float],
  elder[integer], neighbor[integer],
  rand[integer]
- code: do, do*, if [boolean], map
- child:
- type: rand
- name: rand

Inheritance, multi-stack access, subsets
PushGP

Evolves Push programs using (mostly) standard GP.

Multiple types handled without syntactic constraints.

Evolves modules and control structures automatically.
Size Control via Size-Fair Genetic Operators

With Raphael Crawford-Marks, proceedings of *GECCO 2002*.

Table 3: Results for 6-Bit Multiplexor, sorted by computational effort.

<table>
<thead>
<tr>
<th>Crossover Method</th>
<th>Mutation Method</th>
<th>Successful Runs</th>
<th>Average Solution Size</th>
<th>Average Size Limit Replications (Gen. 25)</th>
<th>Average Size Limit Replications (Gen. 49)</th>
<th>Computational Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fair</td>
<td>Fair</td>
<td>30/100</td>
<td>19.80</td>
<td>0.46</td>
<td>28.56</td>
<td>1870000</td>
</tr>
<tr>
<td>Fair</td>
<td>Node Sel</td>
<td>36/100</td>
<td>27.58</td>
<td>71.41</td>
<td>428.67</td>
<td>1885000</td>
</tr>
<tr>
<td>Naive</td>
<td>Fair</td>
<td>32/100</td>
<td>27.53</td>
<td>127.00</td>
<td>410.82</td>
<td>2080000</td>
</tr>
<tr>
<td>Naive</td>
<td>Node Sel</td>
<td>26/100</td>
<td>30.96</td>
<td>389.41</td>
<td>749.47</td>
<td>2520000</td>
</tr>
<tr>
<td>Fair</td>
<td>Naive</td>
<td>26/100</td>
<td>32.27</td>
<td>623.75</td>
<td>1388.20</td>
<td>2635000</td>
</tr>
<tr>
<td>Node Sel</td>
<td>Naive</td>
<td>23/100</td>
<td>37.57</td>
<td>1375.40</td>
<td>1725.29</td>
<td>2835000</td>
</tr>
<tr>
<td>Node Sel</td>
<td>Fair</td>
<td>26/100</td>
<td>27.96</td>
<td>325.13</td>
<td>673.92</td>
<td>3120000</td>
</tr>
<tr>
<td>Naive</td>
<td>Naive</td>
<td>26/100</td>
<td>37.92</td>
<td>972.08</td>
<td>1519.34</td>
<td>3200000</td>
</tr>
<tr>
<td>Node Sel</td>
<td>Node Sel</td>
<td>18/100</td>
<td>31.11</td>
<td>697.06</td>
<td>1014.76</td>
<td>4320000</td>
</tr>
</tbody>
</table>
Evolved Transport Network Agents

Collaboration with Selfridge/Feurzeig/Benyo (MIT/BBN).

Four linked flow corridors per intersection.

“N/S/E/W” arbitrary; nothing rectilinear/2D in underlying network transit simulation.
BBN Transport Network Simulator
Evolved TNAs: Control/Metrics

Agent controls “green time” in one direction.

Metrics available to agent:
- Green time
- Average windowed wait
  - Per corridor
  - “Global”
- Maximum wait
  - Per corridor
  - “Global”
Evolved TNAs: Fitness/Cases

Minimize global wait time.

Average over many different flow density/variability configurations.

<table>
<thead>
<tr>
<th>Fitness Case</th>
<th>North Bound</th>
<th>South Bound</th>
<th>East Bound</th>
<th>West Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>.25</td>
<td>.25</td>
<td>.25</td>
<td>.25</td>
</tr>
<tr>
<td>2</td>
<td>.1</td>
<td>.1</td>
<td>.9</td>
<td>.8</td>
</tr>
<tr>
<td>3</td>
<td>.1</td>
<td>.1</td>
<td>.8</td>
<td>.8</td>
</tr>
<tr>
<td>4</td>
<td>.1</td>
<td>.1</td>
<td>.7</td>
<td>.7</td>
</tr>
<tr>
<td>5</td>
<td>.1</td>
<td>.1</td>
<td>.6</td>
<td>.6</td>
</tr>
<tr>
<td>6</td>
<td>.1</td>
<td>.1</td>
<td>.5</td>
<td>.5</td>
</tr>
<tr>
<td>7</td>
<td>.1</td>
<td>.1</td>
<td>.4</td>
<td>.4</td>
</tr>
<tr>
<td>8</td>
<td>.1</td>
<td>.1</td>
<td>.3</td>
<td>.3</td>
</tr>
<tr>
<td>9</td>
<td>.1</td>
<td>.1</td>
<td>.2</td>
<td>.2</td>
</tr>
<tr>
<td>10</td>
<td>.1</td>
<td>.1</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>11</td>
<td>.9</td>
<td>.9</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>12</td>
<td>.8</td>
<td>.8</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>13</td>
<td>.7</td>
<td>.7</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>14</td>
<td>.6</td>
<td>.6</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>15</td>
<td>.5</td>
<td>.5</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>16</td>
<td>.4</td>
<td>.4</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>17</td>
<td>.3</td>
<td>.3</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>18</td>
<td>.2</td>
<td>.2</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>19</td>
<td>.1</td>
<td>.1</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>20</td>
<td>.9</td>
<td>.9</td>
<td>.1</td>
<td>.1</td>
</tr>
<tr>
<td>21</td>
<td>.3</td>
<td>.3</td>
<td>.5</td>
<td>.5</td>
</tr>
<tr>
<td>22</td>
<td>.9</td>
<td>.01</td>
<td>.01</td>
<td>.01</td>
</tr>
</tbody>
</table>
Evolved TNAs: Agent

NewTimeGreen = OldTimeGreen
+ WindowedAverageWait(northCorridor)
+ WindowedAverageWait(southCorridor)
+ WindowedAverageWait(eastCorridor)
+ WindowedAverageWait(westCorridor)
+ MaxWait(southCorridor)
+ MaxWait(westCorridor)
- MaxWait(northCorridor)
## Evolved TNAs: Performance

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Fitness (summed average wait values across all fitness cases)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolved agent</td>
<td>1.3</td>
</tr>
<tr>
<td>Constant time-green of 0.5</td>
<td>3.1</td>
</tr>
<tr>
<td>Constant time-green of 0.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Constant time-green of 0.8</td>
<td>2.4</td>
</tr>
</tbody>
</table>
Programs evolved in uniformly variable environments were more immediately reactive to changes in their environments.
Evolved “Opera” Agents

Collaboration with Crespi/Cybenko/Russ/Santini (Dartmouth).

Evolve decentralized and coordinated 3D navigation.

Addition of “vector” data type improves performance.

With Alan Robinson, to appear in *Late-Breaking Papers of GECCO 2002.*
Confirmation/Extension of Van Belle/Ackley Effect

Collaboration with Van Belle/Ackley (UNM).

Evolution in a dynamically changing environment ($A\sin(Ax)$, with randomly changing $A$). Modularity allows adaptation via isolation of constant/variable features of the environment.
### Van Belle/Ackley Effect Parameters

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>1000</td>
</tr>
<tr>
<td>Tournament size</td>
<td>5</td>
</tr>
<tr>
<td>Max generations</td>
<td>200</td>
</tr>
<tr>
<td>Fitness cases</td>
<td>50</td>
</tr>
<tr>
<td>Mutation %</td>
<td>45</td>
</tr>
<tr>
<td>Crossover %</td>
<td>45</td>
</tr>
<tr>
<td>Reproduction %</td>
<td>10</td>
</tr>
<tr>
<td>Mutation operators</td>
<td>standard, fair (0.25), perturb (50)</td>
</tr>
<tr>
<td>Crossover operators</td>
<td>standard, fair (0.25), uniform</td>
</tr>
<tr>
<td>Instruction set</td>
<td>ephemeral-random-integer, ephemeral-random-float, ephemeral-random-boolean, ephemeral-random-symbol, convert, =, rep, swap, pop, dup, max, min, &gt;, &lt;, /, <em>, -, +, pulldup, pull, exp, log, cos, sin, not, or, and, nth, list, cons, cdr, car, quote, map, if, do</em>, do, integer, float, boolean, type, code</td>
</tr>
</tbody>
</table>
Van Belle/Ackley Effect Results

![Graph showing the Van Belle/Ackley Effect Results with epochs on the x-axis and average hits of the best-of-generation program on the y-axis. The graph includes two lines: one for epoch start and one for epoch end.](image-url)
Autoconstructive Evolution: Pushpop

Individuals make their own children.

The machinery of reproduction and diversification (and thereby the machinery of evolution) evolves.

Radical self-adaptation.
Adaptive Populations of Pushpop Programs are Reliably Diverse

Partial explanation for emergence of diversifying reproduction in biology.
Breve: a 3D Environment for the Simulation of Decentralized Systems and Artificial Life

Written by Jon Klein, http://www.spiderland.org/breve

Simplifies the rapid construction of complex 3D simulations.

Object-oriented scripting language with rich pre-defined class hierarchy.

OpenGL 3D graphics with lighting, shadows, and reflection.

Rigid body simulation, collision detection/response, articulated body simulation.

Runge-Kutta 4th order integrator or Runge-Kutta-Fehlman integrator with adaptive step-size control.
Breve Swarm

By Jon Klein, after Craig Reynolds.

acceleration = \( p_1 \) *[away from crowding others vector]  
+ \( p_2 \) *[towards world center vector]  
+ \( p_3 \) *[average neighbor velocity vector]  
+ \( p_4 \) *[towards neighbor center vector]  
+ \( p_5 \) *[random vector]
On-Line Evolution of Goal-Directed Swarms

Changes to Breve/swarm:

Multiple species
\[ p_6^* \] [away from crowding other species vector]

Randomly moving energy sources:
\[ p_7^* \] [towards closest energy source vector].

Energy costs:
- Colliding with one another
- Being outnumbered (by species) in neighborhood
- Giving birth
- Surviving (per simulation cycle)

Upon death (energy = 0), parameters replaced with mutated version of fittest of species

Fitness metric = age \* energy
Evolving Goal-Directed Swarms Demo

[“flock nicely” presets, randomize and evolve]
Future

Enhance complexity/realism of environments for agent evolution.

Build capability for evolution of arbitrary (Push) agent programs into 3D Breve environment.

Integrate MIT/BBN elementary adaptive modules into agent evolution system.