Unwitting Distributed Genetic Programming
via Asynchronous JavaScript and XML

Jon Klein and Lee Spector
School of Cognitive Science, Hampshire College

This work was supported by NSF Grant No. 0308540
Introduction

- GP takes time
- Fortunately, GP scales well
- More fitness test evaluations = more results
- Lots of unused computation out there
Distributed GP

- GP is embarrassingly parallel: use more machines for more fitness tests
- Several existing systems/frameworks for distributed GP, including the Distributed Genetic Programming Framework (Weise & Geihs, 2006)
- Existing systems for other evolutionary computing paradigms
Problems with Distributed GP

- Require client-side software installation
- Require client-side motivation
- Require client-side permission
Unwitting Distributed Genetic Programming

- Solve GP problems without (you) running any fitness tests
- All fitness tests run, unwittingly, by unaffiliated web users
- A.K.A. “parasitic computing” — see Nature 412, August 2001
- See also GECCO-2007 workshop paper by Merelo et al.
AJAX

• Asynchronous JavaScript + XML = interactive web applications

• Send data back and forth between client and server from a fully loaded webpage

• Buzzwordy!

• Light-weight, ubiquitous, generally innocuous

• “Web 2.0”: Google Apps, Digg, Amazon use AJAX for interactive web pages
Push3 Language

- Designed for evolutionary computation
- Multi-type stack based language
- Very simple syntax
- Unusually powerful semantics
- Easy to implement
**Push3**

- **KEY IDEA**: Stack-based postfix language with one stack per type: integer, float, vector, Boolean, name, code, exec, ....

- Syntax-independent handling of multiple data types.

- Code and exec stacks support use and evolution of subroutines (any architecture), recursion, evolved control structures, and meta-evolutionary mechanisms.
Push3 Syntax

program ::= instruction | literal | ( program* )
**Push3 Semantics**

- To execute program $P$:
  1. Push $P$ onto the **EXEC** stack.
  2. While the **EXEC** stack is not empty, pop and process the top element of the **EXEC** stack, $E$:
     (a) If $E$ is an instruction: execute $E$ (accessing whatever stacks are required).
     (b) If $E$ is a literal: push $E$ onto the appropriate stack.
     (c) If $E$ is a list: push each element of $E$ onto the **EXEC** stack, in reverse order.
### Sample Push3 Instructions

<table>
<thead>
<tr>
<th>Stack manipulation instructions (all types)</th>
<th>POP, SWAP, YANK, DUP, STACKDEPTH, SHOVE, FLUSH, =</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math (INTEGER and FLOAT)</td>
<td>+, −, /, *, &gt;, &lt;, MIN, MAX</td>
</tr>
<tr>
<td>Logic (BOOLEAN)</td>
<td>AND, OR, NOT, FROMINTEGER</td>
</tr>
<tr>
<td>Code manipulation (CODE)</td>
<td>QUOTE, CAR, CDR, CONS, INSERT, LENGTH, LIST, MEMBER, NTH, EXTRACT</td>
</tr>
<tr>
<td>Control manipulation (CODE and EXEC)</td>
<td>DO*, DO<em>COUNT, DO</em>RANGE, DO*TIMES, IF</td>
</tr>
</tbody>
</table>
A Simple Push3 Program

( 2 3 INTEGER.* 4.1 5.2 FLOAT.+ TRUE FALSE BOOLEAN.OR )

Resulting stacks:

BOOLEAN STACK: ( TRUE )

CODE STACK: ( ( 2 3 INTEGER.* 4.1 5.2 FLOAT.+ TRUE FALSE BOOLEAN.OR ) )

FLOAT STACK: ( 9.3 )

INTEGER STACK: ( 6 )
Resulting stacks:

BOOLEAN STACK: ( TRUE )

CODE STACK: ( ( 4.1 2 ( TRUE ) ( 3 5.2 ( FALSE ) ) FLOAT.+ BOOLEAN.OR INTEGER.* ) )

FLOAT STACK: ( 9.3 )

INTEGER STACK: ( 6 )
You get *ALL* of this for *FREE!* (or at least real cheap)

- Subroutines (with evolved architecture)
- Iterators (standard and evolved)
- Recursion and combinators
- Evolved control structures
- Evolved genetic operators
**PushScript**

- Lightweight (<30k) JavaScript Push implementation
- Supports all standard Push3 stack types, most Push3 instructions
- Runs in most web browsers including Internet Explorer, Firefox, Safari, iPhone*
- Requires NO software installation: loads automatically with webpage

*which is Safari anyway, but it’s just fun to say that our system runs on the iPhone*
Interactive Demo

Type in a Push program below:

```plaintext
(( 5.0 4.0 FLOAT./ 7.0 FLOAT.+ ) ( 2 3 INTEGER.> ))
```

In conjunction with annoyingly buzzwordy AJAX technologies, we can dynamically load a Push program from a server, execute it in a web-browser and submit the results back to the server. Note that this does not require any actual user interaction. It can be done continuously while a user views a webpage.

Run Random Push Program From Server
Server-Side Code

- Lightweight server implementation to avoid server-side bottlenecks
- New fitness cases sent as XML via PHP script
- Data collection via PHP scripts
- New generations generated via breve script, using the C++ Push3 implementation
Process:

1. The browser loads the webpage.
2. The webpage fetches fitness tests and programs via AJAX.
3. The PushScript interpreter runs push fitness tests.
4. The browser loads web content and fitness test server.
5. The PushScript interpreter returns fitness values to the server via AJAX.
6. pushfitnesstest.js
7. push.js
Problems

• 5 simple symbolic regression problems we’ve studied previously

• Deployed on a low traffic website (breve: http://www.spiderland.org/breve)

• Proof of concept question: can unwitting computation be used to solve our GP problems without (us) running fitness tests, and no voluntary user participation?
# Parameters

| Problems                                      | 1. $8 \times x \times x \times x + 3 \times x \times x + x$  
|                                              | 2. $x \times x \times x + x \times x + x$  
|                                              | 3. $x \times x \times x - 2 \times x \times x - x$  
|                                              | 4. $x \times x \times x \times x + x \times x \times x + x \times x + x - 8$  
|                                              | 5. $x \times x \times x \times x \times x - 2 \times x \times x \times x + x \times x - 2$  
| Input \((x)\) values | 1-8  
| Fitness                                      | sum of absolute value of errors  
| Crossover rate                               | 40%  
| Fair mutation rate                           | 40%  
| Deletion mutation rate                       | 5%   
| Duplication rate                             | 15%  
| Population size                              | 2000  
| Maximum program size                         | 50   
| Tournament size                              | 7    
| Ephemeral random constants                   | integers from -10 to 10  
| Instruction set (Dec. 10 problems 1, 2 and 3) | FLOAT.+, FLOAT.-, FLOAT.*, FLOAT./, FLOAT.POP, FLOAT.DUP  
|                                              | FLOAT.SWAP, INPUT  
| Instruction set (Dec. 10 problems 4 and 5, Jan. 15 all) | INTEGER.+ , INTEGER.-, INTEGER.*, INTEGER./, INTEGER.POP,  
|                                              | INTEGER.DUP, INTEGER.SWAP, INPUT |
Results

- Yes! We can solve symbolic regression problems.
- Very, very slowly.
- Several hours to solve a problem which takes a few minutes on the desktop.
- Probably not practical for simple problems, but...
... IT CAN BE PRACTICAL IF:

• ... to compliment local computation on more open-ended problems
• ... local fitness computation takes longer than about .5 seconds per fitness test (on a low-traffic server)
• ... the system is deployed on a very high traffic website
Stealing?

• Short answer, “yes” with an “if”, long answer, “no”, with a “but”

• No more computation than typical AJAX applications

• ... but we’re using the computation for our own benefit
Future Work

• New problem classes: implement domain-specific functions for use with PushScript.
• New problems: which is to say, “real” problems.
• Faster fitness test evaluation: PushJava? PushFlash?