Breaking Browsers: A Survey

Jeffrey Putnam
Kaleb Albee
Eastern Washington University
Genetic Programming

- Encode programs as “genes”
- Evaluate in objective function to get a numeric score
- Usually want to minimize objective function
- Use “crossover” and “mutation” to produce new genes
- Repeat until satisfied
- Usually programs encoded in Lisp-ish syntax
  - (\(\sin (\cos (+ x y)))\))
Stack Based GP

- Genes encoded as strings
- Evaluated in a stack based virtual machine
  - Forth, Postscript...
  - x y + cos sin
- Data held in stack
  - stack data is doubles, strings
- Data held in global state
  - in this case, HTML structure
    - elements, attributes, values...
Genes

• Gene string composed of tokens
• Tokens represent:
  − Strings
    • “foo”
  − Numbers (floating point)
    • (1.345)
  − Functions
    • <element>
• Tokens are atomic
Crossover, Mutation...

• Crossover done by selecting substrings of two genes and pasting together
• Mutation done by copying a gene and deleting, adding and mutating tokens
• Also operators for chopping off beginning and end of genes
• Genes can be generated with non-uniform probabilities for tokens
Evolution

- Start with population
- Evaluate all members of population
- Save best ("elitism")
- Generate unique new genes by selecting genes favoring those that scored well and doing mutation and crossover
- Save a few good genes as a "genepool" to help maintain diversity in future
Project Goals

• Can GP be used to generate debugging data
  – flexible
  – naturally gives “regression testing”
  – but either too easy or too hard
• Originally to just crash browsers
• Used time as a measure
  – because it worked
• Then one page took 3+ hours to display
• Timing became interesting on its own
**HTML data**

- Always a “current element”
- New elements created with an “element” function
  - added as sub-elements to the current element
  - made current element
- Elements may be closed
  - `<foo> .... </foo>`
- or left unclosed
  - `<foo> <bar> </foo>`
- elements generated from list of valid HTML elements and from randomly generated strings
**HTML Data...**

- Attributes defined with an attribute function
- Names, values generated with random strings
  - usually syntactically valid (names may include non-valid characters)
  - but no guarantee
- Attribute values always given as strings
  - `attribute=“value”`
  - even when attribute name is sensible, value usually is not
- Mechanism to limit the length of the generated HTML
Evaluation

• Program runs a tiny “web server”
• Gets request and sends page generated by gene
• Records time sent and time next request received
  - elapsed time is used to compute score
• If browser fails, wrapper restarts requesting page “crash” so evaluator can use fixed time for crash
  - crash recovery time (which may require human intervention) is fixed
  - can be set to reward crashing or not
• Very short times scored the same
Mechanics

- Headers say content is HTML
- Request immediate reload
  - effectively happens after page loads and displays
- Headers also specify length of page (so browsers don't wait for end of malformed HTML)
- Elapsed time measured in milliseconds
- Any other requests generated get zero length responses (images, embedded objects....)
- Socket is closed after content sent
Problems

• Inconsistent browsers
  – some just hang instead of crashing or fetching next page
  – sometimes without using any CPU
  – sometimes pegging CPU at max utilization
  – involves user intervention

• Some browsers don't like being run from scripts

• Getting the headers right to consistently force reloads was involved

• Run “server” and browser on same machine or machines on same local network
Results

• Few browsers are trustworthy
• Most crash for some input – sometimes non-valid, often valid
• For most there is input that can be limited in length but that will cause the browser to take a long, long (long!) time to respond
• Crashing may be results of random search over input space (HTML)
• Clear that long response times are caused by GP engine learning what makes browser slow
Example Log File

- browser = internet-explorer
- normalizing = false
- crash time = 10000
- population = 1000
- generations = 10
- refresh = Refresh: 0;URL="http://146.187.130.103:5555/normal.html"
- truncation = 20000
Example Log File ...

- eval=1043 time=5513 strlen=17133 score=0.01 crash
- eval=1044 time=1012 strlen=7392 score=0.1
- eval=1045 time=1305 strlen=11077 score=0.07692307692307693
- eval=1046 time=286 strlen=9213 score=0.5
• Maximum time in plotted run
  – 4655902 milliseconds
  – 1.3 hours
• crashed at evaluations 5311,7192,8237,8280
• Another run crashed 54 times out of 5000 evaluations
• Crashes occurred in 4 of 6 runs done for this test
• page load times relatively fast until long loading pages dominate
Firefox

- Crashed in 4 of 9 runs
- In one run crashed eventually in 1 out of every four page loads
- Slower page loads
- One page load took 203 minutes to complete
- In one development run (not logged carefully), a single page load took 19 hours
Firefox
Safari

• Load times started slower than IE
  - but remained faster
• Maximum page load time was under four seconds
• Reliable, robust, reasonably fast
Safari
Conclusions

• Genetic Programming has potential to be a debugging/probing tool
• Browsers are nowhere near as robust as they should be in the face of unknown web sites
• Its astonishingly easy to make most browsers run very, very slowly
Potential

• Generate XML according to some schema
  - easy to generate correct syntax
  - much harder to generate correct semantics

• Deserialize XML to objects in some programming language
  - use as part of a test regime
  - ensuring meaningful semantics as most OO code assumes good semantics on created objects