

Applying Genetic Programming to Bytecode and Assembly

Eric Collom

Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

29 April '14,
UMM Senior Seminar



The big picture

- Evolving whole programs is hard to do with source code.
- Evolving whole programs with bytecode and assembly is not as hard.

Outline

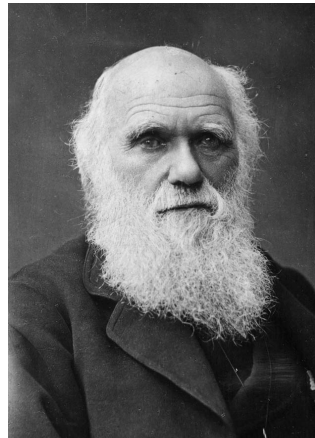
- 1 Background
- 2 Why Evolve Instruction-level Code
- 3 FINCH:Evolving Programs
- 4 Using Instruction-level code to automate bug repair
- 5 Conclusions

Outline

- 1 Background
 - EC
 - Java Bytecode and x86 Assembly
- 2 Why Evolve Instruction-level Code
- 3 FINCH:Evolving Programs
- 4 Using Instruction-level code to automate bug repair
- 5 Conclusions

What is Evolutionary Computation?

- EC is a technique that is used to automate computer problem solving.
- Loosely emulates evolutionary biology.

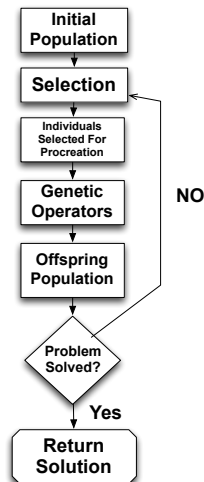


Charles Darwin

<http://tinyurl.com/lqwj3wt>

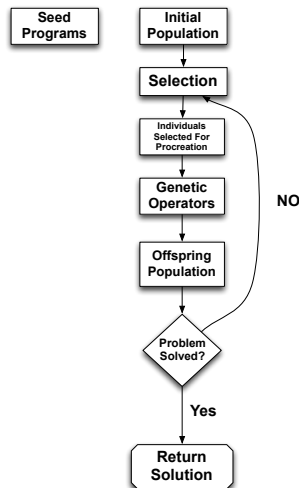
How does it work

- Continuous Optimization
- Selection is driven by the *fitness* of individuals
- Genetic Operators mimic sexual reproduction and mutation



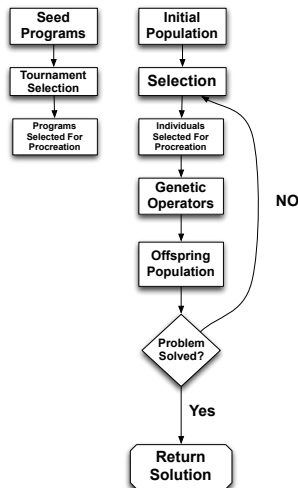
Genetic Programming

- Uses the EC technique to evolve programs
- The population is programs



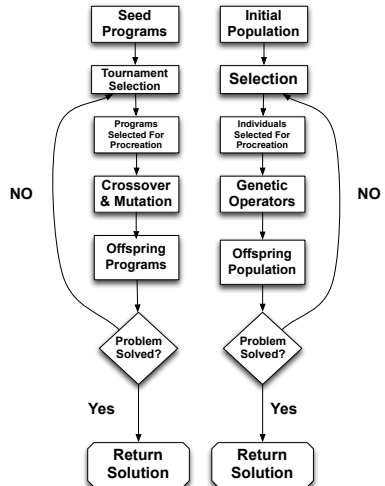
Genetic Programming

■ Tournament Selection

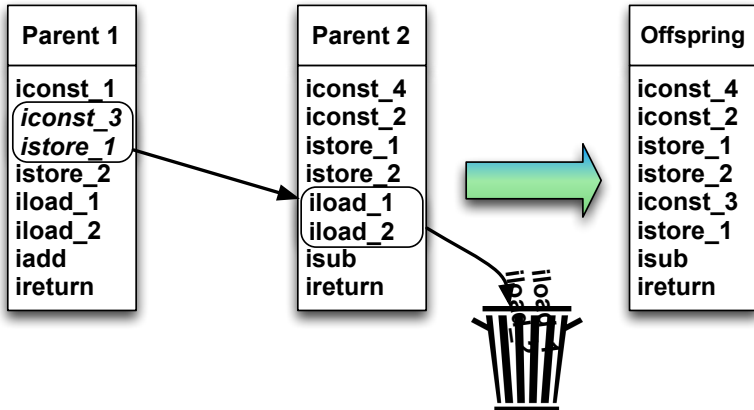


Genetic Programming

- Crossover
- Mutation

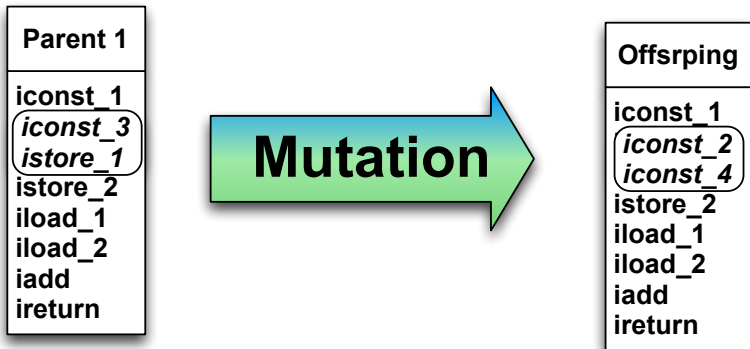


Crossover



Crossover with Java Bytecode

Mutation



Crossover with Java Bytecode

Java Virtual Machine

- Frames
- Array of local variables
- Operand Stack

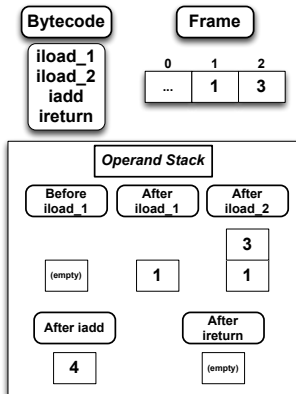


Bytecode

- Opcodes
- Prefix indicates type

iconst_n
istore_n
iload_n
iadd
isub
ireturn

Bytecode and the Stack



How the stack and Java bytecode works

Outline

- 1 Background
- 2 Why Evolve Instruction-level Code
- 3 FINCH:Evolving Programs
- 4 Using Instruction-level code to automate bug repair
- 5 Conclusions

Source Code Constraints

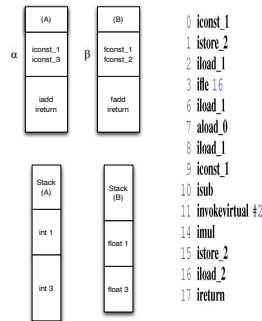
Flexibility

Outline

- 1 Background
- 2 Why Evolve Instruction-level Code
- 3 FINCH:Evolving Programs**
 - How it Works
 - Results
- 4 Using Instruction-level code to automate bug repair
- 5 Conclusions

Selecting Offspring

- There is still a chance to produce non-compilable code
- Solution: Add restrictions to code selection.
- Stack and Frame Depth
- Variable Types
- Control Flow



Crossover

Outline

- 1 Background
- 2 Why Evolve Instruction-level Code
- 3 FINCH:Evolving Programs
- 4 Using Instruction-level code to automate bug repair**
 - How it Works
 - Results
- 5 Conclusions

Selecting Offspring

Genetic Operators

Non-Halting Offspring

Outline

- 1 Background
- 2 Why Evolve Instruction-level Code
- 3 FINCH:Evolving Programs
- 4 Using Instruction-level code to automate bug repair
- 5 Conclusions**

Conclusions

References