Object-Level Recombination of Commodity Applications

Blair Foster          Anil Somayaji

Carleton Computer Security Lab  School of Computer Science
Carleton University

July 10, 2010
The Dream

Wouldn’t it be nice to evolve real programs automatically?
The Reality of “Real Programs”

- **big**
  - millions of lines of code!
- brittle representations
  - C statements versus S-expressions
- brittle semantics
  - complex algorithms and data structures
- fitness functions?!
Hope

Commodity applications—the programs we use every day—can be recombined with members of their own species.

Huh?
Species

Horse × Camel = ∅ ⇒

Mozilla Firefox × Google Chrome = ∅

Firefox 3.6.6 × Firefox 4.0b1 = ???
Mapping Programs to Individuals

executable $\rightarrow$ individual

.o file $\rightarrow$ chromosome

Linking $\rightarrow$ development
Our Contribution

- Recombine program variants represented as object files.
- Parents must share a common recent ancestor.
- Use linker tricks, GA search to find and create viable children.
- Use a weak fitness function that only tests for basic viability; humans choose interesting variants.
ObjRecombGA

[--Target Settings--]
Primary Dir:
Secondary Dir:
Target Binary:
Primary Build String:
Secondary Build String:
Common Build String:
Object Restrict List:
Test Script:

[✔] Avoid Convergence
[✔] Force Test

[--General Settings--]
Scripts Dir: /home/blair/Masters/svn/192.168.2.239/masters/code/scripts/
Recombination Dir: /home/blair/Masters/TEMP/working/
Temp Dir: /home/blair/Masters/TEMP/
Population Size: 10  # Generations: 5
Selection Algorithm: TournamentSelection
X-over Algorithm: SinglePointXover
Tournament Prob.: 0.90  Elite #: 0  Diversity Max Prob.: 0.90
Initial Population List:

<[ Start ]>
Interesting GA characteristics

- Bitstring representation: each bit selects an object file
- Small populations: 12-50 individuals
- Few generations: 20 or less
- Fast convergence to maximum fitness
  - Coarse fitness function that gives points for compiling, not crashing, and a few basic functionality tests.

Remember, this GA is designed to explore, not optimize!
The Linking Problem
The Linking Solution

- Allow linking against all object files: bitstring indicates linking priority.
- Do our own symbol resolution to choose appropriate variant:
  - Catalog symbols and precompute potential dependency chains.
  - Mangle all symbols to make them unique, remember mappings.
  - Rewrite symbol references to refer to appropriate version of code or data.
- And, search for viable offspring using a GA!
Quake: HUD, no models
Quake: Fisheye
Quake: Recombined Child
Quake: Parents and Children
Fitness Convergence

Note that populations exhibit high diversity.
Limitation of current implementation

- Two ancestors only.
- Child programs cannot be used as ancestors
  - Children are effectively diploid, parents are haploid.
- Linking sometimes fails when it could have succeed.
- Primitive fitness functions.
Potential Applications

- Software testing/debugging
- Functional diversity for security
- User-directed software evolution
Conclusion

- Commodity software can be recombined using object files as the units of recombination.
- Linking problems are significant but can be overcome with linker tricks.
- Functional dependencies are addressed through GA search.
- Only works with closely related programs.
- Defines a “species” relationship amongst programs.
- Opens the door to automated evolution of the software we use everyday.
Acknowledgements

- Funded by Canada’s NSERC Discovery Program
- Blair Foster (Master’s student!)
- “Road to Software Evolvability,” Santa Fe Institute 2005
- Stephanie Forrest (graduate advisor), David Ackley

Contact me at soma@ccsl.carleton.ca

Questions?