My Interdisciplinary Trajectory

Arizona State U., 2017 - present

University of New Mexico, 1990 - 2017

Santa Fe Institute, 1990 - present

Center for Nonlinear Studies, LANL, 1988 - 1990

Univ. of Michigan, 1985

St. John’s College, 1977
The Role of NSF

• Lucky breaks
  – PYI Award letter “Computational aspects of the immune system” (1991)
  – Interdisciplinary research becomes socially acceptable
  – The web
• 30 years of NSF funding, rarely large grants
• Goal for talk: Make the case for strong connections between biology and computation, beyond neurons
The Biology of Computation

• Defending complex systems from malicious behavior
  – Vaccine design, cancer, other evolving pathogens
  – Ch 1. Computer immune systems

• Engineering and evolution of software
  – Ch 2. Micro-level: Evolutionary computation methods
  – Ch 3. Macro-level: Inadvertent evolution

Computer Immune System  Evolving Software

networkworld.com
Information Processing in the Immune System

- Learned distinction between self and other
- Primary response to new foreign antigen
- Evolved biases towards common pathogens

- Secondary response
- Cross-reactive memory

- $10^{11} - 10^{16}$ different foreign patterns from ~25,000 genes

Edward Jenner’s first smallpox vaccine performed on James Phipps in 1796

http://www.history.com/news/vaccines diseases forgotten
Cybersecurity Recapitulates Biology

- Anomaly intrusion detection, signature detection
- Address space randomization
- Natural diversity for N-variant systems
- Two-factor authentication
- Ratchets, constructive neutral evolution
- Limits to defense-in-depth?

Hochberg et al. *Nature*, 2020
Evolution in Software

- Macro-level: Inadvertent evolution
- Micro-level: Evolutionary computation methods
Micro-evolution of Software

The Secret Sauce

• Start with a working program
• Mutations mimic human operations
  – Delete, Copy, Move/Replace
  – Don’t invent new code, statement-level operations
• Restrict mutations to statements executed by failing test cases
• Most bugs are small

```c
void zunebug_repair(int days) {
    int year = 1980;
    while (days > 365) {
        if (isLeapYear(year)) {
            // days -= 366; // repair delete
            year += 1;
        }
        else {
            days -= 366; // repair insert
        }
    }
    else {
        days -= 365;
        year += 1;
    }
    printf("current year is %d\n", year);
}
```
How well does it work in practice?

- Large systematic empirical studies with many tools
  - Defects4J: Java programs (all tools---36% correct)
  - ManyBugs: Large opensource C programs (72% plausible)
- Industry transitions
- The Machine Learning tsunami
- Caveats
  - Buggy test cases
  - “Overfitting” (patch vs. repair)
  - What is a correct repair?
  - Assumptions made by tool, e.g., fault localization to a single line
  - Reproducibility
  - Difficult to know what the ML models have been trained on
Dorsa Amir @DorsaAmir · 2m
When your code is a mess but it somehow still works.
Biological Properties of Software

Eric Schulte, Joe Renzullo, Jhe-Yu Liou

• Mutational robustness
  – Mutation testing considered helpful
• Neutral landscapes
• Fitness distributions
  – Where should we look for repairs?
• Epistasis (interactions among genes)
Neutral Mutations

• Many biological mutations leave fitness unchanged
  – Buffering, genetic potential
• A neutral mutation passes the original test suite
  – It may or may not pass held-out failing test cases
  – Plentiful: ~30% of GenProg mutations are neutral!

```java
if (right > left) {
  // code elided
  quick(left, r)
  quick(l, right)
}
```

Neutral Mutations Enable Search

- Engineered diversity
- Reducing energy consumption
- For bug repairs
- For reducing GPU run-times
Neutral Landscapes

- Neutral mutations sometimes repair latent bugs
- Many semantically distinct repairs
  - Color indicates unique repairs
- Network connects diverse repairs by neutral intermediate mutations
- Insight: All repairs are neutral wrt original test suite
Fitness Distributions:
Where are the repairs in neutral space?

1. Generate large pool of neutral edits
2. Generate random subsets of pool
3. Apply each subset to original program
4. Measure repair frequency

100 times more likely to find a patch at distance 200 than at distance 1
Evolving Faster GPU Code

*J. Liou, C. Wu and S. Forrest (TACO, 2020)*

- GPUs important for ML and HPC, but challenging to optimize
- More complex mutation operators
- 49% average speedup on Rodinia benchmarks (NVIDIA Tesla P100)

**Optimizations:** Application logic, architecture-specific, dataset specific
Optimizing Multiple Sequence Alignment Codes
(J. Liou, M. Gul Awan, C. Wu, S. Hofmeyr, and S. Forrest, ISWC 2022)

- Smith-Waterman algorithm (ADEPT)
  - State-of-the art implementation on GPU
  - Hand-optimized for GPU by human expert
- GEVO run
  - 256 pop size; 300 gens; 7 days
  - 64 mutations, 17 useful
  - 5 independent mutations (7%)
  - 12 interdependent (18% improvement)

GEVO finds 28.5% run-time improvement over expert human-optimized version
GEVO optimizations are epistatic

- Rearrange usage of sub-memory systems on GPU (15%)
  - Use shared memory instead of private registers
- Remove redundant synchronizations (~4%)
  - violates CUDA Programming guide
- Remove unnecessary memory initializations (30X on adept-b)

Epistatic optimizations can be hard for humans to find
The Bigger Picture

• Key ingredients of Darwinian evolution
  – Variation: Mutation and recombination
  – Natural selection
  – Inheritance

• Software
  – Selection and inheritance: Successful genes are copied: libraries, packages, code snippets, etc.
  – Variation: Programmers make small changes and recombine successful genes

_Thesis: Software today is the result of many generations of inadvertent evolution_
Macro-evolution in Software

Continuous Integration

Stack overflow + github

Uber Two-factor authentication attack
Arms races
# The Tinkerer and the Craftsman

<table>
<thead>
<tr>
<th>Evolution</th>
<th>Engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unplanned and openended</td>
<td>Planned, with specifications</td>
</tr>
<tr>
<td>Survival, relative fitness</td>
<td>Purposeful, goal-driven</td>
</tr>
<tr>
<td>Ongoing process</td>
<td>Clean slate design</td>
</tr>
<tr>
<td>Incremental</td>
<td>Large jumps</td>
</tr>
<tr>
<td>Driven by random mutation</td>
<td>Conducted by agents with foresight and intent</td>
</tr>
</tbody>
</table>

‘Nature is a tinkerer, not an inventor’

F. Jacob
Evolution and Engineering

- Antibiotic resistance
- Directed evolution
- Synthetic biology
- Attack fuzzing in cybersecurity
- Large jumps in evolution
- Randomized algorithms
- Software
Evolution and Engineering

- Antibiotic resistance
- Directed evolution
- Synthetic biology
- Attack fuzzing in cybersecurity
- Large jumps in evolution
- Randomized algorithms
- Software
Evolution and Engineering

- Antibiotic resistance
- Directed evolution
- Synthetic biology, xenobots
- Attack fuzzing in cybersecurity
- Large jumps in evolution
- Randomized algorithms
- Software
Evolution and Engineering

- Antibiotic resistance
- Directed evolution
- Synthetic biology
- Attack fuzzing in cybersecurity
- Large jumps in evolution
- Randomized algorithms
- Software
What are the best practices for engineering systems in the context of evolution?

- Claim: Software is an excellent starting point
- Co-evolution
  - Interactions with humans
  - Interactions among software components
  - Interactions with biology
- Highly optimized tolerance
  - Understanding tradeoffs between performance and robustness (Carlson and Doyle)
- Rethinking defense-in-depth and technological ratchets
Summary

• The perspective of biology is important because it provides insight and guidance
  – Engineering (bio-inspired computing)
  – Science (biological properties of computation)

"As engineers, we would be foolish to ignore the lessons of a billion years of evolution”

Carver Mead
THANK YOU

steph@asu.edu

https://profsforrest.github.io
References