Neutral Networks of Real-World Programs and their Application to Automated Software Evolution

Eric Schulte

Department of Computer Science
University of New Mexico
Albuquerque, NM

July 2014
Software Engineering

Motivation

Bugs cost estimated $312 billion per year
[Britton, University of Cambridge 2013]
Software Engineering

Motivation

**Bugs** cost estimated $312 billion per year
[Britton, University of Cambridge 2013]

**Security** millions of dollars per exploit
[Van, OECD Publishing 2008]
Software Engineering

Motivation

**Bugs** cost estimated $312 billion per year
[Britton, University of Cambridge 2013]

**Security** millions of dollars per exploit
[Van, OECD Publishing 2008]

**Resources** 1% of global energy to data centers in 2010
[Koomey, Analytics Press 2011]
Software Engineering

Motivation

**Bugs**  cost estimated $312 billion per year  
[Britton, University of Cambridge 2013]

**Security**  millions of dollars per exploit  
[Van, OECD Publishing 2008]

**Resources**  1% of global energy to data centers in 2010  
[Koomey, Analytics Press 2011]

**Developers**  projected 1.5 million in US by 2018  
[U.S Department of Labor, 2011]
Evolved Ecosystem

- applications
- libraries
- compilers
- operating systems
- architectures
Software Engineering
Evolution and Automation

Evolved Ecosystem
- applications
- libraries
- compilers
- operating systems
- architectures

Evolvable Software
- automated diversity
- assembly and binary repair
- patch closed source executable
- optimization
Genprog
Automatically Fix Bugs in C Software

Collaboration between UNM and UVA

Dr. Stephanie Forrest

Dr. Westley Weimer
Genprog
Automatically Fix Bugs in C Software

Input
 ✓✓✓✓ X

Evolve
Fitness Evaluation

Mutants

Output
 ✓✓✓✓✓

Accept
Minimize
Discard

[Le Goues, 2013, Figure 3.2]
Genprog
Automatically Fix Bugs in C Software

Strengths

Effective  Repaired 55/105 bugs for $8 each
General   Multiple classes of bugs and security defects
Best Papers ICSE 2009, GECCO 2009, SBST 2009
Humies    Gold 2009, Bronze 2012
Genprog
Automatically Fix Bugs in C Software

But
Why doesn’t this break my software?
Outline

Introduction

Program Representation and Transformation

Mutational Robustness and Neutral Networks

Application: Program Diversity

Application: Assembler- and Binary-Level Program Repair

Application: Patching Closed Source Executables

Application: Optimizing nonfunctional Program Properties

Future Work

Conclusion
Program Representation

Source

```c
if (a==0){
    printf("%g\n", b); }
else {
    while (b!=0){
        if(a>b){ a=a-b; }
        else { b=b-a; }}
    printf("%g\n", a);
```
Program Representation

Source

```c
if (a == 0) {
    printf("%g\n", b); }
else {
    while (b != 0) {
        if (a > b) { a = a - b; }
        else { b = b - a; }
    }
    printf("%g\n", a);
}
```

Tree
Program Representation

Source

```c
if (a == 0) {
    printf("%g\n", b);
} else {
    while (b != 0) {
        if (a > b) {
            a = a - b;
        } else {
            b = b - a;
        }
    }
    printf("%g\n", a);
}
```

Tree

```
if(a==0)
    printf("%g\n",b);
else
    while(b!=0)
        if(a>b)
            a=a-b;
        else
            b=b-a;
printf("%g\n",a);
```

Vector

```
main:
    .cfi_startproc
    pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq %rsp, %rbp
    .cfi_def_cfa_register 6
    subq $48, %rsp
```
Program Representation

Representations

<table>
<thead>
<tr>
<th>Rep.</th>
<th>Type</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLang</td>
<td>tree</td>
<td>C-family source</td>
</tr>
<tr>
<td>CIL</td>
<td>tree</td>
<td>C Int. Lang.</td>
</tr>
<tr>
<td>LLVM</td>
<td>vector</td>
<td>LLVM SSA IR</td>
</tr>
<tr>
<td>ASM</td>
<td>vector</td>
<td>x86 &amp; ARM</td>
</tr>
<tr>
<td>ELF</td>
<td>vector</td>
<td>x86 &amp; MIPS</td>
</tr>
</tbody>
</table>

Tree: **CIL, CLang**

```
if(a==0)
  printf("%g\n",b);
while(b!=0)
  if(a>b)
    a=a-b;
  else
    b=b-a;
printf("%g\n",a);
```

Vector: **ASM, LLVM, ELF**

```
main:
  .cfi_startproc
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset 6, -16
  movq %rsp, %rbp
  .cfi_def_cfa_register 6
  subq $48, %rsp
```
Program Mutation Operations

Tree Delete: 

Vector Delete: 

Tree Insert: 

Vector Insert: 

Tree Swap: 

Vector Swap:
Program Mutation Operations

Tree Delete

Tree Insert

Tree Swap

ELF Delete

Vector Insert

Vector Swap
Program Mutation Operations

Tree Delete

Tree Insert

Tree Swap

ELF Delete

ELF Replace

Vector Swap
C source

```c
int main(int argc, char *argv[]){
    int x=2;
    x+=3;
    x=x*x;
    x=x*x;
    printf("%d\n", x);
    return 0;
}
```
LLVM

%\text{x} = \text{alloca} \ i32, \ \text{align} \ 4
\text{store} \ i32 \ 2, \ i32* \ %\text{x}, \ \text{align} \ 4
%0 = \text{load} \ i32* \ %\text{x}, \ \text{align} \ 4
%\text{add} = \text{add} \ \text{nsw} \ i32 \ %0, \ 3
\text{store} \ i32 \ %0, \ i32* \ %\text{x}, \ \text{align} \ 4
%1 = \text{load} \ i32* \ %\text{x}, \ \text{align} \ 4
%2 = \text{load} \ i32* \ %\text{x}, \ \text{align} \ 4
%\text{mul} = \text{mul} \ \text{nsw} \ i32 \ %1, \ %2
\text{store} \ i32 \ %\text{mul}, \ i32* \ %\text{x}, \ \text{align} \ 4
%3 = \text{load} \ i32* \ %\text{x}, \ \text{align} \ 4
%\text{call} = \text{call} \ ... \ \text{printf}(\text{\text{\@str} \ %3})
LLVM Mutation Illustration

LLVM: Delete 4

%\texttt{x} = \texttt{alloca i32, align 4}
\texttt{store i32 2, i32* \%x, align 4}
%\texttt{0} = \texttt{load i32* \%x, align 4}
%\texttt{add} = \texttt{add nsw i32 \%0, 3}
\texttt{store i32 \%0, i32* \%x, align 4}
%\texttt{1} = \texttt{load i32* \%x, align 4}
%\texttt{2} = \texttt{load i32* \%x, align 4}
%\texttt{mul} = \texttt{mul nsw i32 \%1, \%2}
\texttt{store i32 \%mul, i32* \%x, align 4}
%\texttt{3} = \texttt{load i32* \%x, align 4}
%\texttt{call} = \texttt{call ... @printf(@.str \%3)}
LLVM Mutation Illustration

LLVM: Delete 4

%\texttt{x} = \texttt{alloca i32}, \texttt{align 4}
\texttt{store i32 2, i32* \texttt{x}, align 4}
%\texttt{0} = \texttt{load i32* \texttt{x}, align 4}
%\texttt{add} = \texttt{add nsw i32 %0, 3}
\texttt{store i32 %0, i32* \texttt{x}, align 4}
%\texttt{1} = \texttt{load i32* \texttt{x}, align 4}
%\texttt{2} = \texttt{load i32* \texttt{x}, align 4}
%\texttt{mul} = \texttt{mul nsw i32 %1, %2}
\texttt{store i32 %mul, i32* \texttt{x}, align 4}
%\texttt{3} = \texttt{load i32* \texttt{x}, align 4}
%\texttt{call} = \texttt{call ... @printf(@.str %3)}
LLVM Mutation Illustration

**LLVM: Insert 9 4**

\[
\begin{align*}
%x &= \text{alloca i32, align 4} \\
\text{store i32 2, i32* %x, align 4} \\
%0 &= \text{load i32* %x, align 4} \\
%\text{add} &= \text{add nsw i32 %0, 3} \\
\text{store i32 %add, i32* %x, align 4} \\
%1 &= \text{load i32* %x, align 4} \\
%2 &= \text{load i32* %x, align 4} \\
%\text{mul} &= \text{mul nsw i32 %1, %2} \\
\text{store i32 %mul, i32* %x, align 4} \\
%3 &= \text{load i32* %x, align 4} \\
%\text{call} &= \text{call ... @printf(@.str %3)}
\end{align*}
\]
LLVM Mutation Illustration

LLVM: Insert 9 4

\%
x = alloca i32, align 4
store i32 2, i32* \%x, align 4
\%0 = load i32* \%x, align 4
\%add = add nsw i32 \%0, 3
store i32 \%add, i32* \%x, align 4
\%1 = load i32* \%x, align 4
\%2 = load i32* \%x, align 4
\%mul = mul nsw i32 \%1, \%2
\%add.insert = add nsw i32 \%0, 3
store i32 \%mul, i32* \%x, align 4
\%3 = load i32* \%x, align 4
\%call = call ... @printf(@.str \%3)
LLVM: Insert 9 4

%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
%add.insert = add nsw i32 %0, 3
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
## Conclusion

### Representations

<table>
<thead>
<tr>
<th>Rep.</th>
<th>Type</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLang</td>
<td>tree</td>
<td>C-family source</td>
</tr>
<tr>
<td>CIL</td>
<td>tree</td>
<td>C Int. Lang.</td>
</tr>
<tr>
<td>LLVM</td>
<td>vector</td>
<td>LLVM SSA IR</td>
</tr>
<tr>
<td>ASM</td>
<td>vector</td>
<td>x86 &amp; ARM</td>
</tr>
<tr>
<td>ELF</td>
<td>vector</td>
<td>x86 &amp; MIPS</td>
</tr>
</tbody>
</table>

### Differences

- Mutation operations and Search Space
- Languages
- Requirements
- Expression as executables
- Communication of mutations
Outline

Introduction

Program Representation and Transformation

Mutational Robustness and Neutral Networks

Application: Program Diversity

Application: Assembler- and Binary-Level Program Repair

Application: Patching Closed Source Executables

Application: Optimizing nonfunctional Program Properties

Future Work

Conclusion
Software Mutational Robustness

[Schulte et al., GPEM 2013]

percentage of mutants which are functional
Software Mutational Robustness

Program Space

Original Program

Specification

Test Suite

Neutral Mutant

Killed Mutant

mutants which are functional

Software Mutational Robustness

Neutral Networks and Automated Software Evolution

Mutational Robustness and Neutral Networks
Software Mutational Robustness

Program Space

Specification

Test Suite

Original Program

Neutral Mutant

Neutral Networks and Automated Software Evolution
Mutational Robustness and Neutral Networks
Software Mutational Robustness

Program Space

Specification

Test Suite

Original Program

Neutral Mutant

Killed Mutant

Neutral Networks and Automated Software Evolution
Mutational Robustness and Neutral Networks
Definition

\[ \text{MutRB}(P, T, M) = \left| \left\{ P' \mid m \in M, P' \leftarrow m(P) \land T(P') \right\} \right| \]

- \( P \) \hspace{0.5cm} \text{program}
- \( T \) \hspace{0.5cm} \text{test suite}
- \( M \) \hspace{0.5cm} \text{mutation operators}
Mutational Robustness by Test Suite

**Sorters**

- bubble-sort
- insertion-sort
- merge-sort
- quick-sort

**Siemen’s**

- printtokens
- schedule
- sed
- space
- tcas

**Systems Programs**

- bzip2 1.0.2
- ccrypt 1.2
- imagemagick 6.5.2
- jansson 1.3
- lighttpd 0.5.0
- nullhttpd 0.5.0
- oggenc 1.0.1
- potion 40b5f03
- redis 3.2.8
- tiff 3.8.2
- vyquon 335426d
- grep

---

Neutral Networks and Automated Software Evolution

Mutational Robustness and Neutral Networks
Fitness Distribution by Representation

![Fitness Distribution by Representation](image_url)

- CLANG
- CIL
- LLVM
- ASM
- ELF

Neutral Networks and Automated Software Evolution
Fitness Landscapes and Neutral Networks

Program Space

- Specification
- Test Suite
- Original Program
- Neutral Mutant
- Killed Mutant
Fitness Landscapes and Neutral Networks
Fitness Landscapes and Neutral Networks

Program Space

Test Suite

Specification

Original Program

Neutral Mutant

Killed Mutant

Fitness

Neutral Networks and Automated Software Evolution

Mutational Robustness and Neutral Networks
Fitness Landscapes and Neutral Networks

![Diagram of Fitness Landscapes and Neutral Networks]

- Program Space
- Test Suite
- Fitness

Neutral Networks and Automated Software Evolution
Mutational Robustness and Neutral Networks
Fitness Landscapes and Neutral Networks

Fitness Landscapes and Neutral Networks
Fitness Landscapes and Neutral Networks
Fitness Landscapes and Neutral Networks
Span of Neutral Networks

Insertion Sort Neutral Variants

Avg. LOC

Number of Applied Mutations

Avg. LOC

% Neutral Variants

% Neutral Variants

Avg. LOC

0 50 100 150 200 250

0 5 10 15 20 25

12 13 14 15 16 17 18 19 20 21

170 180 190 200 210 220 230

Neutral Networks and Automated Software Evolution
Mutational Robustness and Neutral Networks
Span of Neutral Networks

Insertion Sort Neutral Variants: size controlled

![Graph showing the relationship between the number of applied mutations and average LOC and percentage of neutral variants.](Image)
Random Walks in Program Space

Quick Sort Random Variants

Fraction of Variants which are Neutral

Number of Applied Mutations

Fraction Found Neutral Experimentally

Neutral Networks and Automated Software Evolution

Mutational Robustness and Neutral Networks
Conclusion

Software Neutral Networks are

- **wide** span large distances in program space
- **thick** consistent mutational robustness
- **accessible** automatically explorable
Conclusion

Software Neutral Networks are

- **wide** span large distances in program space
- **thick** consistent mutational robustness
- **accessible** automatically explorable
Outline

Introduction

Program Representation and Transformation

Mutational Robustness and Neutral Networks

**Application: Program Diversity**

Application: Assembler- and Binary-Level Program Repair

Application: Patching Closed Source Executables

Application: Optimizing nonfunctional Program Properties

Future Work

Conclusion
Automated Diversity

Program Space

Specification

Test Suite

Original Program
Automated Diversity

Program Space

Specification

Test Suite

Original Program

Mutant

Neutral Networks and Automated Software Evolution
Application: Program Diversity
Automated Diversity

Program Space

Specification

Test Suite
Automated Diversity

Program Space

Specification

Test Suite

Neutral Networks and Automated Software Evolution
Application: Program Diversity
Automated Diversity
Proactive Bug Repair

Program Space

Specification

Test Suite

Bug

Original Program
Mutant
Neutral Networks and Automated Software Evolution
Application: Program Diversity
Contribution

Automated Diversity

- Neutral networks expose meaningful diversity
  [Baudry, ISSTA 2014]
- Diverse variants proactively fix defects
- Select small mutually diverse populations
Program Repair in Embedded Devices
[Schulte et al., ASPLOS 2013]

Resource Constraints
- Small disks
- Less memory
- Slow processors
- Slow, costly comm.
Genprog
Automatically Fix Bugs in C Software

Strengths

**Effective**  Repaired 55/105 bugs for $8 each

**General**  Multiple classes of bugs and security defects

**Best Papers**  ICSE 2009, GECCO 2009, SBST 2009

**Humies**  Gold 2009, Bronze 2012
Genprog
Automatically Fix Bugs in C Software

Strengths

- **Effective** Repaired 55/105 bugs for $8 each
- **General** Multiple classes of bugs and security defects
- **Best Papers** ICSE 2009, GECCO 2009, SBST 2009
- **Humies** Gold 2009, Bronze 2012

Requirements

- Source code
- Build tool chain
- Expensive fault localization
- Expensive fitness function (compilation, test execution)
Genprog: Assembler- and Binary-Level
Automatically Fix Bugs in C Software

Strengths

- **Effective**  Repaired 55/105 bugs for $8 each
- **General**  Multiple classes of bugs and security defects
- **Best Papers**  ICSE 2009, GECCO 2009, SBST 2009
- **Humies**  Gold 2009, Bronze 2012

Requirements

- **Source code**
- **Build tool chain**
- **Expensive fault localization**
- **Expensive fitness function** (compilation, test execution)
## Embedded Repair Benchmark Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Program Description</th>
<th>Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>atris</td>
<td>graphical tetris game</td>
<td>stack buffer exploit</td>
</tr>
<tr>
<td>ccrypt</td>
<td>encryption utility</td>
<td>segfault</td>
</tr>
<tr>
<td>deroff</td>
<td>document processing</td>
<td>segfault</td>
</tr>
<tr>
<td>flex</td>
<td>lexical analyzer generator</td>
<td>segfault</td>
</tr>
<tr>
<td>indent</td>
<td>source code processing</td>
<td>infinite loop</td>
</tr>
<tr>
<td>look svr4</td>
<td>dictionary lookup</td>
<td>infinite loop</td>
</tr>
<tr>
<td>look ultrix</td>
<td>dictionary lookup</td>
<td>infinite loop</td>
</tr>
<tr>
<td>merge</td>
<td>merge sort</td>
<td>duplicate inputs</td>
</tr>
<tr>
<td>merge-cpp</td>
<td>merge sort (in C++)</td>
<td>duplicate inputs</td>
</tr>
<tr>
<td>s3</td>
<td>sendmail utility</td>
<td>buffer overflow</td>
</tr>
<tr>
<td>uniq</td>
<td>duplicate text processing</td>
<td>segfault</td>
</tr>
<tr>
<td>units</td>
<td>metric conversion</td>
<td>segfault</td>
</tr>
<tr>
<td>zune</td>
<td>embedded media player</td>
<td>infinite loop</td>
</tr>
</tbody>
</table>
Contribution

ASM and ELF representation

- Reduce requirements of automated repair
  - Effective: fixing 12 (ASM) and 11 (ELF) of 13
  - 62% faster runtime
  - 95% smaller disk footprint
  - 86% less memory
- Reduce resources needed to perform repair
- New program repair search spaces
Outline

Introduction

Program Representation and Transformation

Mutational Robustness and Neutral Networks

Application: Program Diversity

Application: Assembler- and Binary-Level Program Repair

Application: Patching Closed Source Executables

Application: Optimizing nonfunctional Program Properties

Future Work

Conclusion
NETGEAR Exploit
[Schulte et al., unpublished]

1. URL starting with BRS bypasses authentication
2. URL including unauth.cgi or securityquestions.cgi bypass authentication
3. unprotected page removes authentication for every page
   http://router/BRS_02_genieHelp.html
Common Vulnerability

NETGEAR WNDR4700

D-Link DIR-100

Linksys WAG200G
NETGEAR Repair Overview

Neutral Networks and Automated Software Evolution

Application: Patching Closed Source Executables
NETGEAR Repair Overview

Repair Runtime

evaluations $\sim 36,000$
runtime 86.6 min.
threads 32
Contribution

Genprog Requirements

- Non-stripped ELF file
- Regression test suite
- Fault localization
Contribution

Genprog Requirements

- Non-stripped ELF file
- Regression test suite
- Fault localization
Contribution

Genprog Requirements

▶ Non-stripped ELF file
▶ Regression test suite
▶ Fault localization

Enables

▶ repair of black box executables
▶ repair in data sections
▶ patching proprietary software
Outline

Introduction

Program Representation and Transformation

Mutational Robustness and Neutral Networks

Application: Program Diversity

Application: Assembler- and Binary-Level Program Repair

Application: Patching Closed Source Executables

Application: Optimizing nonfunctional Program Properties

Future Work

Conclusion
Optimization Problem
[Schulte, Dorn et al., ASPLOS 2014]

Optimizing complex non-functional properties

\[ \text{properties} \times \text{hardware} \times \text{environment} \]

- properties: memory, network, energy, etc.
- hardware: architectures, processors, memory stack, etc.
- environment: variables, load, etc.

Every program transformation requires

- a-priori reasoning
- manual implementation
- guaranteed correctness
Genetic Optimization Algorithm (GOA)

Post-compiler, test-driven, Genetic Optimization Algorithm

Post-compiler

source → GCC → .s → GOA → .s → .exe
Genetic Optimization Algorithm (GOA)

Post-compiler, test-driven, Genetic Optimization Algorithm

Test driven

Use test cases to exercise program

- evaluate functionality
- measure runtime properties
Genetic Optimization Algorithm (GOA)

Post-compiler, test-driven, Genetic Optimization Algorithm

Algorithm

Assembler → Fitness Function → Workload
Fitness Function → Population
Population → Mutate → Executable

Minimize → Profile → Fitness
Fitness
Genetic Optimization Algorithm (GOA)

Post-compiler, test-driven, Genetic Optimization Algorithm

Algorithm

Assembler ➔ Fitness Function ➔ Workload

Fitness Function ➔ Population ➔ Mutate ➔ Profile

Minimize ➔ Executable

Fitness = \( 2^{10} \)

evals = \( 2^{18} \)

time = \( \sim 16h \)
## Benchmark Applications

<table>
<thead>
<tr>
<th>Program</th>
<th>C/C++ Lines of Code</th>
<th>ASM Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>510</td>
<td>7,932</td>
<td>Finance modeling</td>
</tr>
<tr>
<td>bodytrack</td>
<td>14,513</td>
<td>955,888</td>
<td>Human video tracking</td>
</tr>
<tr>
<td>facesim</td>
<td></td>
<td></td>
<td>no alternate inputs</td>
</tr>
<tr>
<td>ferret</td>
<td>15,188</td>
<td>288,981</td>
<td>Image search engine</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>11,424</td>
<td>44,681</td>
<td>Fluid dynamics animation</td>
</tr>
<tr>
<td>freqmine</td>
<td>2,710</td>
<td>104,722</td>
<td>Frequent itemset mining</td>
</tr>
<tr>
<td>raytrace</td>
<td></td>
<td></td>
<td>no testable output</td>
</tr>
<tr>
<td>swaptions</td>
<td>1,649</td>
<td>61,134</td>
<td>Portfolio pricing</td>
</tr>
<tr>
<td>vips</td>
<td>142,019</td>
<td>132,012</td>
<td>Image transformation</td>
</tr>
<tr>
<td>x264</td>
<td>37,454</td>
<td>111,718</td>
<td>MPEG-4 video encoder</td>
</tr>
<tr>
<td>total</td>
<td>225,467</td>
<td>1,707,068</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** PARSEC benchmark applications.
Hardware Platforms

AMD Server

Intel Desktop

Neutral Networks and Automated Software Evolution
Application: Optimizing nonfunctional Program Properties
Results: Energy Reduction

![Energy Reduction Chart]

- `<blacksholes>`
- `<bodytrack>`
- `<ferret>`
- `<fluidanimate>`
- `<freqmine>`
- `<swaptions>`
- `<vips>`
- `<x264>`

**Average Energy Reduction:**

- **AMD**
- **Intel**

---

Neutral Networks and Automated Software Evolution

Application: Optimizing nonfunctional Program Properties
## Functionality on Withheld Tests

<table>
<thead>
<tr>
<th>Program</th>
<th>AMD</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>bodytrack</td>
<td>92%</td>
<td>100%</td>
</tr>
<tr>
<td>ferret</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>6%</td>
<td>31%</td>
</tr>
<tr>
<td>freqmine</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>swaptions</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>vips</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>x264</td>
<td>27%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Anecdotes
Blackscholes

- 90% less energy
- removed redundant outer loop
- modified semantics
Anecdotes

Blackscholes
► 90% less energy
► removed redundant outer loop
► modified semantics

Swaptions
► 42% less energy
► improved branch prediction
► hardware specific
Anecdotes

Blackscholes

- 90% less energy
- removed redundant outer loop
- modified semantics

Swaptions

- 42% less energy
- improved branch prediction
- hardware specific

Vips

- 20% less energy
- substitution of memory access for calculation
- resource trade-off
Contribution

Genetic Optimization Algorithm

1. optimize complex runtime properties (energy)
2. leverages particulars of hardware, and environment
3. reveal compiler inefficiencies
4. find efficiencies, e.g., loop elimination
5. transformations presented as ASM diff
Outline

Introduction

Program Representation and Transformation

Mutational Robustness and Neutral Networks

Application: Program Diversity

Application: Assembler- and Binary-Level Program Repair

Application: Patching Closed Source Executables

Application: Optimizing nonfunctional Program Properties

Future Work

Conclusion
Challenges

Interface
- Guarantee semantics preservation
- Communicate goals (fitness functions)
- Incorporate into development cycle

Power
- Novel functionality
- Combine components
Future Work

Verification, Mutation, Crossover, Hardening, Fitness-Distance Corr.

- Assembler diff chunks formally equivalent
- Static analysis to,
  - find bugs
  - measure code quality
- Invariant preservation
e.g., Daikon
Future Work

Verification, Mutation, Crossover, Hardening, Fitness-Distance Corr.

“Smarter” mutation operations

- Repair templates and external code libraries
- Type-aware operations
- Learned mutation heuristics
Future Work

Verification, Mutation, Crossover, Hardening, Fitness-Distance Corr.

Heterologous Crossover
between different
- compilation flags
- versions
- implementations
- programs
Future Work

Verification, Mutation, Crossover, Hardening, Fitness-Distance Corr.

Future Work 42
Future Work

Verification, Mutation, Crossover, Hardening, Fitness-Distance Corr.

Augment tests to increase granularity

- Count unique values of program counter
- Real-valued oracle comparison (e.g., numdiff)
Outline

Introduction
Program Representation and Transformation
Mutational Robustness and Neutral Networks
Application: Program Diversity
Application: Assembler- and Binary-Level Program Repair
Application: Patching Closed Source Executables
Application: Optimizing nonfunctional Program Properties
Future Work

Conclusion
Extension software is:

1. Mutationally robust with large neutral networks
Conclusion

Extant software is

1. Mutationally robust with large neutral networks
2. A product of evolution
Conclusion

Extant software is

1. Mutationally robust with large neutral networks
2. A product of evolution
3. Amenable to automated evolutionary improvement
Conclusion

**Extant software is**

1. Mutationally robust with large neutral networks
2. A product of evolution
3. Amenable to automated evolutionary improvement

**Software evolution might**

facilitate the automation of software engineering
Thank You

Eric Schulte

email eschulte@cs.unm.edu
homepage https://cs.unm.edu/~eschulte
dissertation https://cs.unm.edu/~eschulte/dissertation
References


Neutral Networks and Automated Software Evolution  References 46
Backup Slides

- Program Space
- Program Expression
- Many Bugs
- LLVM SSA
- Robustness and Adaptation Time
- Proactive Bug Repair
- Fault Localization
- Embedded Repair Results
- Distributed Repair

- Benefits of ASM
- Embedded Repair Space
- Profiling (GOA)
- Minimization (GOA)
- Energy Model (GOA)
- Runtime and Energy Reduction (GOA)
- Physical Evolutionary Computation
Program Space

Total

$$n_{i+1} = \frac{n_i(d - i)(e - 1)}{(i + 1)}$$

Unique

$$n_{z,i+1} = \frac{n_{z-1,i}(d - i)1 + n_z,i(d - i)(e - 2)}{i + 1}$$

Neutral

$$A_i = \sum_{Z \leq i} n_{z,i} \binom{i}{z}^{-1}$$

$$R = \sum_{i \leq d} r^i \sum_{Z \leq i} n_{z,i} \binom{i}{z}^{-1}$$

Constants: \(d=3, e=5\)

<table>
<thead>
<tr>
<th>Step</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Program Space

Back

Total

\[ n_{i+1} = \frac{n_i(d - i)(e - 1)}{(i + 1)} \]

Unique

\[ n_{z,i+1} = \frac{n_{z-1,i}(d - i)1 + n_{z,i}(d - i)(e - 2)}{i + 1} \]

Neutral

\[ A_i = \sum_{z \leq i} n_{z,i} \left( \begin{array}{c} l \\ z \end{array} \right)^{-1} \]

\[ R = \sum_{i \leq d} r^i \sum_{z \leq i} n_{z,i} \left( \begin{array}{c} l \\ z \end{array} \right)^{-1} \]

Constants: \( d = 3, e = 5 \)

\[
\begin{array}{c|c}
\text{Step} & \text{Total} \\
0 & 1 \\
1 & 12 \\
2 & \\
3 & \\
\end{array}
\]
Program Space

**Total**

\[ n_{i+1} = \frac{n_i (d - i) (e - 1)}{(i + 1)} \]

**Unique**

\[ n_{z,i+1} = \frac{n_{z-1,i} (d - i) 1 + n_{z,i} (d - i) (e - 2)}{i + 1} \]

**Neutral**

\[ A_i = \sum_{z \leq i} n_{z,i} \binom{i}{z}^{-1} \]

\[ R = \sum_{i \leq d} r^i \sum_{z \leq i} n_{z,i} \binom{i}{z}^{-1} \]

Constants: \( d=3, e=5 \)

<table>
<thead>
<tr>
<th>Step</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Neutral Networks and Automated Software Evolution  Backup Slides
Program Space

Back

Total

\[ n_{i+1} = \frac{n_i (d - i)(e - 1)}{(i + 1)} \]

Unique

\[ n_{z,i+1} = \frac{n_{z-1,i}(d - i)1 + n_{z,i}(d - i)(e - 2)}{i + 1} \]

Neutral

\[ A_i = \sum_{z \leq i} n_{z,i} \binom{l}{z}^{-1} \]

\[ R = \sum_{i \leq d} r^i \sum_{z \leq i} n_{z,i} \binom{l}{z}^{-1} \]

Constants: \( d=3, \ e=5 \)

<table>
<thead>
<tr>
<th>Step</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
</tr>
</tbody>
</table>
Program Expression

Neutral Networks and Automated Software Evolution
Systematic Evaluation

[Le Goues et al., ICSE 12] Back

Amazon EC2 Servers

Version Control Repositories

- fbc
- gmp
- gzip
- libtiff
- ...

GenProg
## Fixing Bugs for $8 a Bug

<table>
<thead>
<tr>
<th>Program</th>
<th>Defects Repaired</th>
<th>Avg. Cost per Non-Repair Hours</th>
<th>Avg. Cost Per Repair Hours</th>
<th>US$</th>
<th>US$</th>
</tr>
</thead>
<tbody>
<tr>
<td>fbc</td>
<td>1 / 3</td>
<td>8.52</td>
<td>5.56</td>
<td>6.52</td>
<td>4.08</td>
</tr>
<tr>
<td>gmp</td>
<td>1 / 2</td>
<td>9.93</td>
<td>6.61</td>
<td>1.60</td>
<td>0.44</td>
</tr>
<tr>
<td>gzip</td>
<td>1 / 5</td>
<td>5.11</td>
<td>3.04</td>
<td>1.41</td>
<td>0.30</td>
</tr>
<tr>
<td>libtiff</td>
<td>17 / 24</td>
<td>7.81</td>
<td>5.04</td>
<td>1.05</td>
<td>0.04</td>
</tr>
<tr>
<td>lighttpd</td>
<td>5 / 9</td>
<td>10.79</td>
<td>7.25</td>
<td>1.34</td>
<td>0.25</td>
</tr>
<tr>
<td>php</td>
<td>28 / 44</td>
<td>13.00</td>
<td>8.80</td>
<td>1.84</td>
<td>0.62</td>
</tr>
<tr>
<td>python</td>
<td>1 / 11</td>
<td>13.00</td>
<td>8.80</td>
<td>1.22</td>
<td>0.16</td>
</tr>
<tr>
<td>wireshark</td>
<td>1 / 7</td>
<td>13.00</td>
<td>8.80</td>
<td>1.23</td>
<td>0.17</td>
</tr>
</tbody>
</table>

**Total**  
55 / 105  
11.22h  
1.60h
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
%.x = alloca i32, align 4
store i32 2, i32* %.x, align 4
%.0 = load i32* %.x, align 4
%.add = add nsw i32 %.0, 3
store i32 %.add, i32* %.x, align 4
%.1 = load i32* %.x, align 4
%.2 = load i32* %.x, align 4
%.mul = mul nsw i32 %.1, %.2
store i32 %.mul, i32* %.x, align 4
%.3 = load i32* %.x, align 4
%.call = call ... @printf(@.str %.3)
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
Robustness and Adaptation Time

$K = 5, P = 100$

$K = 30, P = 100$

$K = P = 100$

[Draghi et al., Nature 2010]
Proactive Bug Repair

Figure: Bugs Fixed by Number of Variants in Potion
Fault Localization

Focus Mutation

Fault Localization

1. instrument
2. compile
3. evaluate

```c
if(a==0)
    puts(b);
else
    if(a>b)
        a=a-b;
    else
        b=b-a;
while(b!=0)
    if(a>b)
        puts(a);
    else
        b=b-a;
```

Neutral Networks and Automated Software Evolution  Backup Slides  55
Fault Localization

Focus Mutation
Fault Localization
1. instrument
2. compile
3. evaluate
Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.

CPU

Machine-code Instructions

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```
Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.

Machine-code Instructions:

```assembly
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```
Light Weight Fault Localization

1. Sample program counter.

2. Translate memory addresses to program offsets.

3. Smooth sample with Gaussian convolution.

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```
**Light Weight Fault Localization**

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```
Embedded Repair Results

- Effective
  - 62% faster runtime
  - 95% smaller disk footprint
  - 86% less memory

### Total bugs repaired

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>13</td>
</tr>
<tr>
<td>ASM</td>
<td>12</td>
</tr>
<tr>
<td>ELF</td>
<td>11</td>
</tr>
</tbody>
</table>

### Average success rate

100 runs per bug

<table>
<thead>
<tr>
<th>Rep.</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>78.17%</td>
</tr>
<tr>
<td>ASM</td>
<td>70.75%</td>
</tr>
<tr>
<td>ELF</td>
<td>65.83%</td>
</tr>
</tbody>
</table>
Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

### Expected fitness evaluations

<table>
<thead>
<tr>
<th></th>
<th>Rep.</th>
<th>Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>583.98</td>
<td></td>
</tr>
<tr>
<td>ASM</td>
<td>188.38</td>
<td></td>
</tr>
<tr>
<td>ELF</td>
<td>207.15</td>
<td></td>
</tr>
</tbody>
</table>

### Total runtime

<table>
<thead>
<tr>
<th></th>
<th>Rep.</th>
<th>Sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>229.50</td>
<td></td>
</tr>
<tr>
<td>ASM</td>
<td>278.30</td>
<td></td>
</tr>
<tr>
<td>ELF</td>
<td>74.20</td>
<td></td>
</tr>
</tbody>
</table>
Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Example: Merge Sort

Repair by representation

- AST, 2 of 4900 Swaps
- ASM, 1 of 280 Deletes

merge.c

```c
if(left[l-mid-1] <= right[0]){
    result = list; 
}
else{ /* fix: swap branches */
    result = merge(left,l-mid, 
                   right,mid); }
```

merge.s

```assembly
    cmpl %eax, %edx ; fix: del.
    jg .L12
    movq -72(%rbp), %rax
```
## Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

### Disk size

<table>
<thead>
<tr>
<th>Rep.</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>AST</td>
<td>Source code &amp; build toolchain</td>
</tr>
<tr>
<td>ASM</td>
<td>Assembly code &amp; linker</td>
</tr>
<tr>
<td>ELF</td>
<td>Compiled executable</td>
</tr>
</tbody>
</table>
Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

<table>
<thead>
<tr>
<th></th>
<th>MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rep.</td>
<td></td>
</tr>
<tr>
<td>AST</td>
<td>1402</td>
</tr>
<tr>
<td>ASM</td>
<td>756</td>
</tr>
<tr>
<td>ELF</td>
<td>200</td>
</tr>
</tbody>
</table>
Distributed Embedded Repair

Back
Distributed Embedded Repair
Distributed Embedded Repair
Distributed Embedded Repair
### Distributed Genetic Repair Evaluation

#### Relative performance of Distributed Algorithm

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Expected Fitness Evaluations</th>
<th>Wall Clock Seconds</th>
<th>w/SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.94</td>
<td>0.89</td>
<td>1.07</td>
</tr>
<tr>
<td>3</td>
<td>0.84</td>
<td>0.67</td>
<td>0.81</td>
</tr>
<tr>
<td>4</td>
<td>0.80</td>
<td>0.55</td>
<td>0.63</td>
</tr>
</tbody>
</table>
### Relative performance of Distributed Algorithm

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Expected Fitness Evaluations</th>
<th>Wall Clock Seconds</th>
<th>w/ SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.94</td>
<td>0.89</td>
<td>1.07</td>
</tr>
<tr>
<td>3</td>
<td>0.84</td>
<td>0.67</td>
<td>0.81</td>
</tr>
<tr>
<td>4</td>
<td>0.80</td>
<td>0.55</td>
<td>0.63</td>
</tr>
</tbody>
</table>
## Distributed Genetic Repair Evaluation

### Relative performance of Distributed Algorithm

<table>
<thead>
<tr>
<th># Nodes</th>
<th>Expected Fitness Evaluations</th>
<th>Wall Clock Seconds</th>
<th>w/ SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.94</td>
<td>0.89</td>
<td>1.07</td>
</tr>
<tr>
<td>3</td>
<td>0.84</td>
<td>0.67</td>
<td>0.81</td>
</tr>
<tr>
<td>4</td>
<td>0.80</td>
<td>0.55</td>
<td>0.63</td>
</tr>
</tbody>
</table>
The ASM representation performs well.

Avoids
- direct addresses
- argumented instructions

Similarities to DNA
- Linear vector genome
- read sequentially
- reading frames
- start and stop codes
- padding
- bootstrapped compilers
Embedded Repair Search Space

Program size
\[ \approx 3 \times \text{more assembly instructions than C statements} \]

Search space size
\[ = |\text{alphabet}|^{\text{program size}} \]

Possible program coverage

- Possible Programs
- Reachable Programs
- Original Program
Hardware Performance Counters
$ perf stat -- ./blackscholes 1 input /tmp/output

6,864,315,342 cycles
5,062,293,918 instructions
2,944,060,039 r533f00
1,113,084,780 cache-references
  1,122,960 cache-misses

3.227585368 seconds time elapsed
## Delta Debugging

<table>
<thead>
<tr>
<th>Assembly Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addl %ebx, %ecx</code></td>
<td>Location: L808</td>
</tr>
<tr>
<td><code>addl %ebx, %ecx</code></td>
<td>Location: L808</td>
</tr>
<tr>
<td><code>.byte 0x33</code></td>
<td>Location: L970</td>
</tr>
<tr>
<td><code>addq %rdx, %r14</code></td>
<td>Location: L970</td>
</tr>
<tr>
<td><code>xorpd %xmm1, %xmm7</code></td>
<td>Location: L970</td>
</tr>
<tr>
<td><code>cmpq %r13, %rdi</code></td>
<td>Location: L970</td>
</tr>
<tr>
<td><code>cmpl %ecx, %esi</code></td>
<td>Location: L970</td>
</tr>
</tbody>
</table>

**Assembler Fitness Function Workload**

**Minimize Population Profile**

**Neutral Networks and Automated Software Evolution**

**Backup Slides**
### Delta Debugging

<table>
<thead>
<tr>
<th>Line</th>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>5358</td>
<td>c5358</td>
<td><code>addl %ebx, %ecx</code></td>
</tr>
<tr>
<td>5416</td>
<td>c5416</td>
<td><code>addl %ebx, %ecx</code></td>
</tr>
<tr>
<td>5463</td>
<td>c5463</td>
<td><code>addq %rdx, %r14</code></td>
</tr>
<tr>
<td>5841</td>
<td>d5839</td>
<td><code>addq %rdx, %r14</code></td>
</tr>
<tr>
<td>6309</td>
<td>c6307</td>
<td><code>xorpd %xmm1, %xmm7</code></td>
</tr>
<tr>
<td>6413</td>
<td>a6412</td>
<td><code>cmpl %ecx, %esi</code></td>
</tr>
</tbody>
</table>
Minimization

Back

Delta Debugging

```assembly
5358 c5358
<.L808:
   ---
   > addl %ebx, %ecx

5416 c5416
< addl %ebx, %ecx
   ---
   > .L808:

5463 c5463
< .L970:
   ---
   > .byte 0x33

5651 d5650
< .loc 1 457 0 is_stmt 0 discriminator 2

5841 d5839
< addq %rdx, %r14

6309 c6307
< xorpd %xmm1, %xmm7
   ---
   > cmpq %r13, %rdi

6413 a6412
> cmpl %ecx, %esi
```
Minimization

Delta Debugging

Neutral Networks and Automated Software Evolution
Energy Model

Energy Model

energy \frac{time}{cycle} = C_{\text{const}} + C_{\text{ins}} \frac{ins}{cycle} + C_{\text{flops}} \frac{flops}{cycle} + C_{\text{tca}} \frac{tca}{cycle} + C_{\text{mem}} \frac{mem}{cycle}

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Description</th>
<th>Intel (4-core)</th>
<th>AMD (48-core)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_{\text{const}}</td>
<td>constant power draw</td>
<td>31.530</td>
<td>394.74</td>
</tr>
<tr>
<td>C_{\text{ins}}</td>
<td>instructions</td>
<td>20.490</td>
<td>-83.68</td>
</tr>
<tr>
<td>C_{\text{flops}}</td>
<td>floating point ops.</td>
<td>9.838</td>
<td>60.23</td>
</tr>
<tr>
<td>C_{\text{tca}}</td>
<td>cache accesses</td>
<td>-4.102</td>
<td>-16.38</td>
</tr>
<tr>
<td>C_{\text{mem}}</td>
<td>cache misses</td>
<td>2962.678</td>
<td>-4209.09</td>
</tr>
</tbody>
</table>

Table: Energy model coefficients.
GOA: Runtime and Energy Reduction

Backup Slides 65
Physical Evolutionary Computation

Subpopulation (64 individuals)

Hardware Tile
- 70MHz ARM7TDMI MCU
- 32KB RAM, 16KB EEPROM
- 128KB problem data storage
- hardware timers
- power sharing, 4 serial ports

Software Process
∀ evolutionary tile
randomly create an initial population
for ever, concurrently
with frequency \( f_m \)
select one individual and mutate
with frequency \( f_c \)
select two individuals and crossover
with frequency \( f_s \)
select one individual and share
with frequency \( f_d \) if collecting data
report statistics to central collector

Computational Space
Sample geometry: Sixteen 4.75\( \times \)4.75 \( cm \) tiles arranged in a 4 \( \times \) 4 square.

Problem Space
Sample fitness function: Gravity data from the state of Montana. Each hardware tile covers 25\( \times \)25\( \text{'} \) latitude \times \text{longitude} mapped using the ‘local disjoint’ spatial mapping. The size and shape of the total covered area is determined by the number and configuration of the hardware tiles, and by the scale and scheme used to map the hardware space to the problem space.