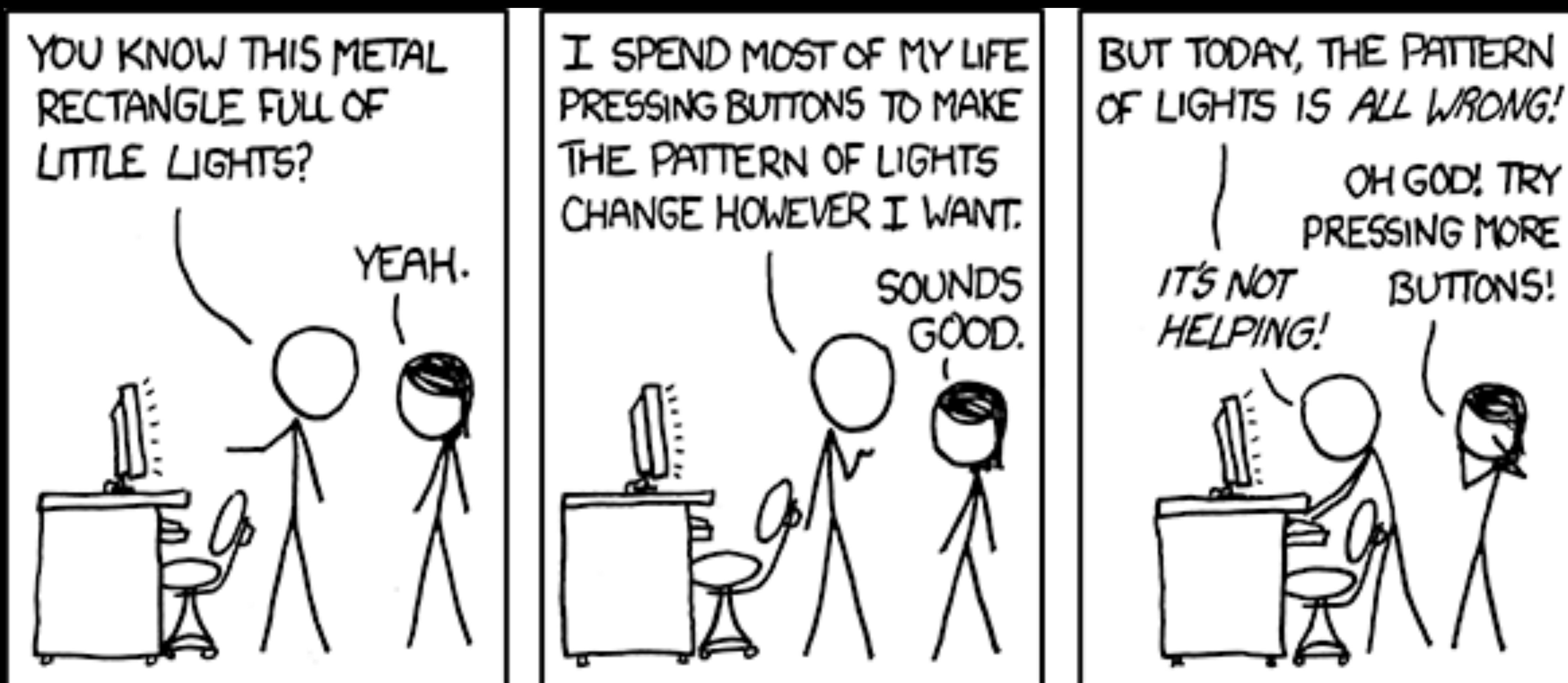
Designing Better Fitness Functions for Automatic Program Repair

Ethan Fast
Claire Le Goues
Stephanie Forrest
Westley Weimer

UVA & UNM


YOU KNOW THIS METAL RECTANGLE FULL OF LITTLE LIGHTS?
YEAH.
I SPEND MOST OF MY LIFE PRESSING BUTTONS TO MAKE THE PATTERN OF LIGHTS CHANGE HOWEVER I WANT.
SOUNDS GOOD.
BUT TODAY, THE PATTERN OF LIGHTS IS ALL WRONG!
OH GOD! TRY PRESSING MORE BUTTONS!
IT'S NOT HELPING!

Motivation

- Software maintenance: **.5**% of U.S. GDP
- How to reduce debugging cost?
  - Evolutionary Software Repair
  - ICSE 2009, GECCO 2010
- Potential Difficulties
  - Test suites dominate execution time
  - Fitness function precision

Presentation Outline

- Review: Program repair via Genetic Programming
- Fitness Function Efficiency
  - Test Suite Sampling
  - Evaluating performance gains
- FItness Function Precision and Smoothness
  - Dynamic Predicates
  - Evaluating Repair Evolution
- Conclusions

GP Program Repair

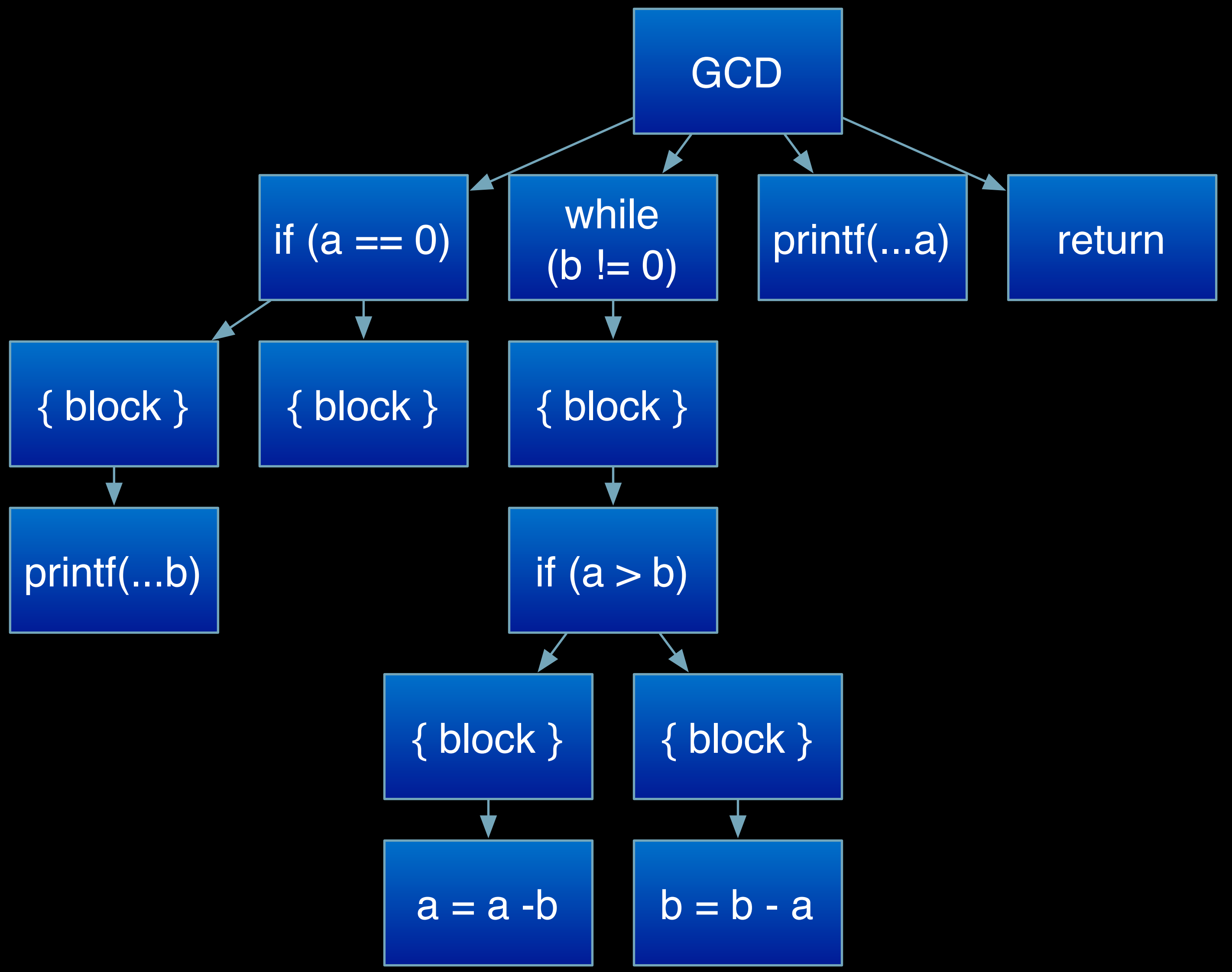
- program source code
- regression tests
- test case illustrating bug

- generate program variants
- run them on test cases
  - *collect predicate information*
  - *select test cases*
- repeat with selection, crossover, mutation

- new program that passes test cases
- or, no solution

Representation

- Individuals represented as *Abstract Syntax Trees*
- Weighted Path
    - Genetic operations occur along stmts executed on failed run
- Only use stmts from other parts of the program



GP Program Repair Details

- To compute fitness, compile a variant
  - If it fails to compile, then fitness = 0
  - Otherwise, run test cases
  - Now, fitness = # tests passed
  - Negative test case(s) more heavily weighted

Issue 1: Slow Test Cases

- A bottleneck in the GP repair process
  - Time spent compiling and executing test cases far exceeds that spent on the GP algorithm
- An impediment to scalability
  - Larger programs and more complex bugs
  - Require larger regression test suites
- Test suite selection and reduction

Test Suite Reduction

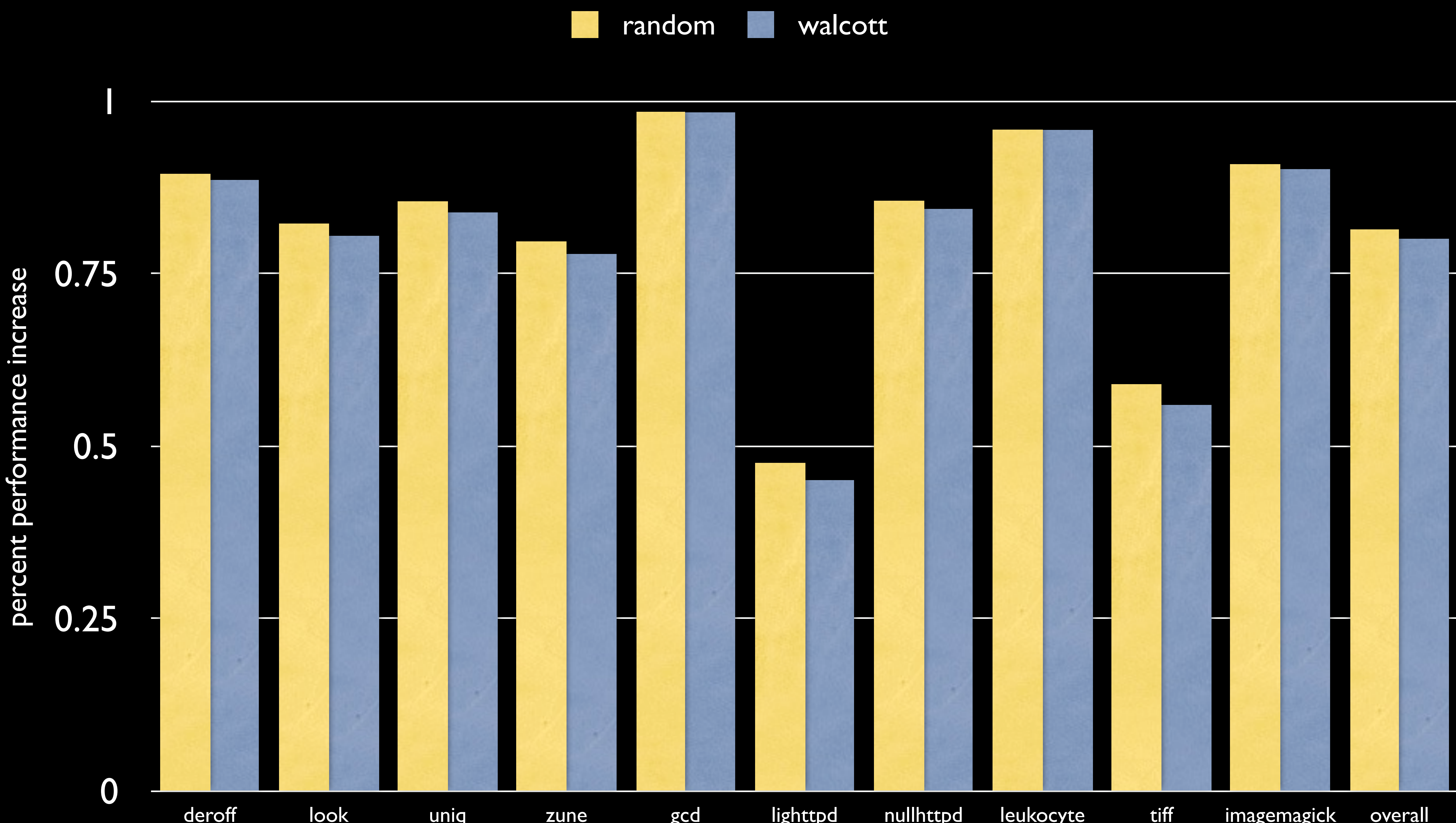
- Idea: Evaluate variants on small random subset of test cases
- Correctness preserving final test: if a variant passes its subset, then evaluate it on the entire suite
- More efficient, reduces the cost of each fitness evaluation
- Introduces noise, potentially "leading the search astray"
  - The benefit of a cheaper fitness function outweighs the negative impact of additional noise
  - E.g. *leukocyte* repair reduced from 90 to 6 min.

Experimental Setup

- Questions
  - Effect of test case sampling on GP repair performance?
  - Does noise outweigh the savings in evaluation time?
- Metric: Average test case evaluations per successful repair
- Two Algorithms
  - *Random*: selects a test suite subset at random
  - *Walcott*: test suite reduction based on genetic algorithms*
- Evaluation: performance gains relative to full test suite

* K. Walcott and M.L. Soffa and G. Kapfhammer and R. Roos. /Time-Aware Test Suite Prioritization/. ISSTA, pp. 1-12, 2006.

Experimental Results



Findings

- Sampling reduces repair time by 81%
  - 10 programs repaired in an average of 1.8 minutes
- Performance gains scale linearly with the percentage of the test suite unsampled, up to 98%
- Random outperforms Walcott
- Two outliers
  - *lighttpd* and *tiff*: ease of repair reduced potential for improvement

Explaining Efficiency Gains

- Extra Experiments
  - 1) Gains from dependent test cases?
  - 2) Similarity of "parent" and "child" fitness?
- Results
  - Ruled out (1) and (2)
  - GP tolerates the noise introduced by test suite reduction

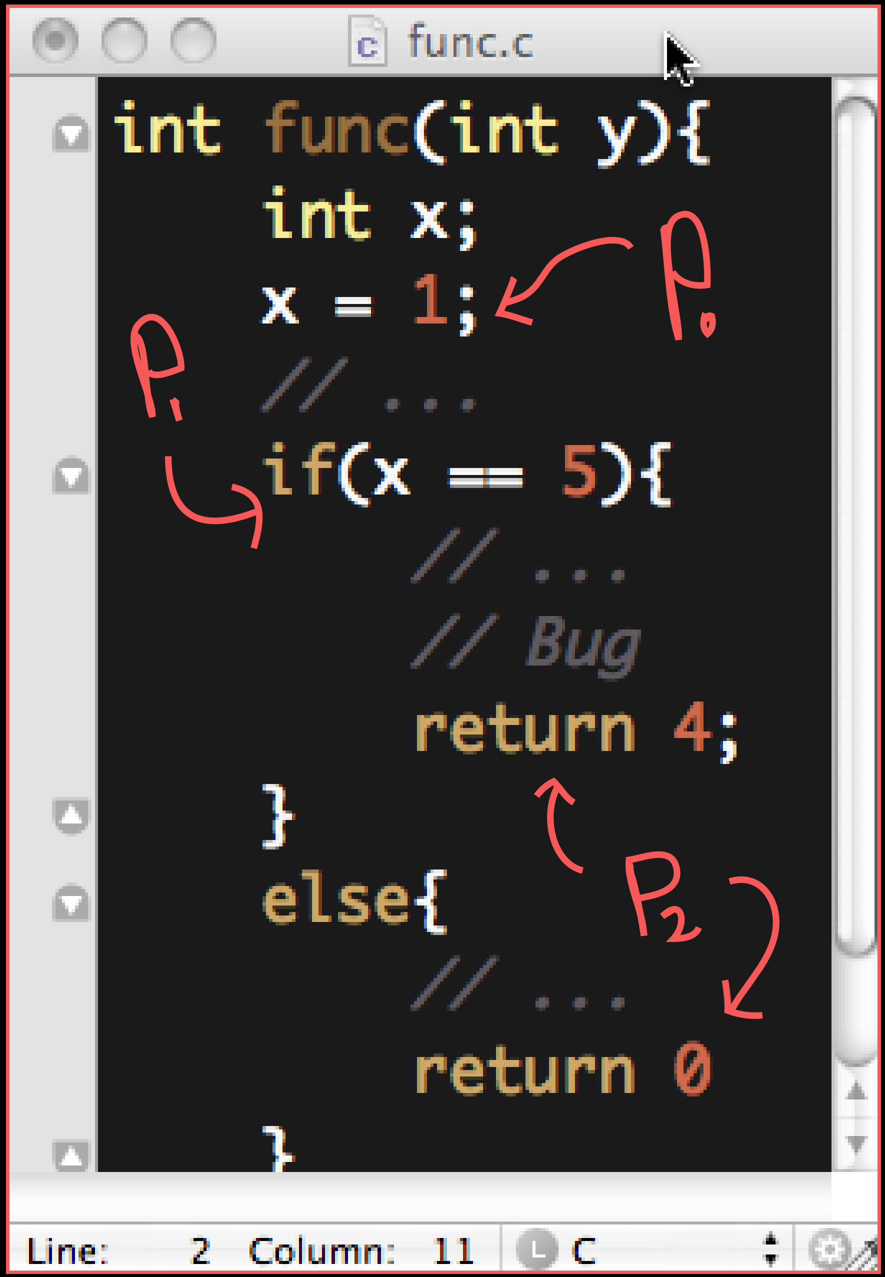
Issue 2: Fitness Precision

- Test cases exhibit all-or-nothing behavior => 7 FF values
- Partial solutions are difficult to reward
  - Required for more complex bugs
  - Ex: a program missing lock and unlock; adding one w/o other could produce lower fitness
- Idea: Finer grain fitness function using dynamic predicates
  - Collect truth values about program statements (e.g. branch conditions, variable assignments) over a run

Predicate Example

	Pass T.C.	Fail T.C.
P_0	x ? y	x ? y
P_1	TRUE	FALSE
P_2	0	4

- P_0: scalar pairs on x
- P_1: branch condition
- P_2: return values



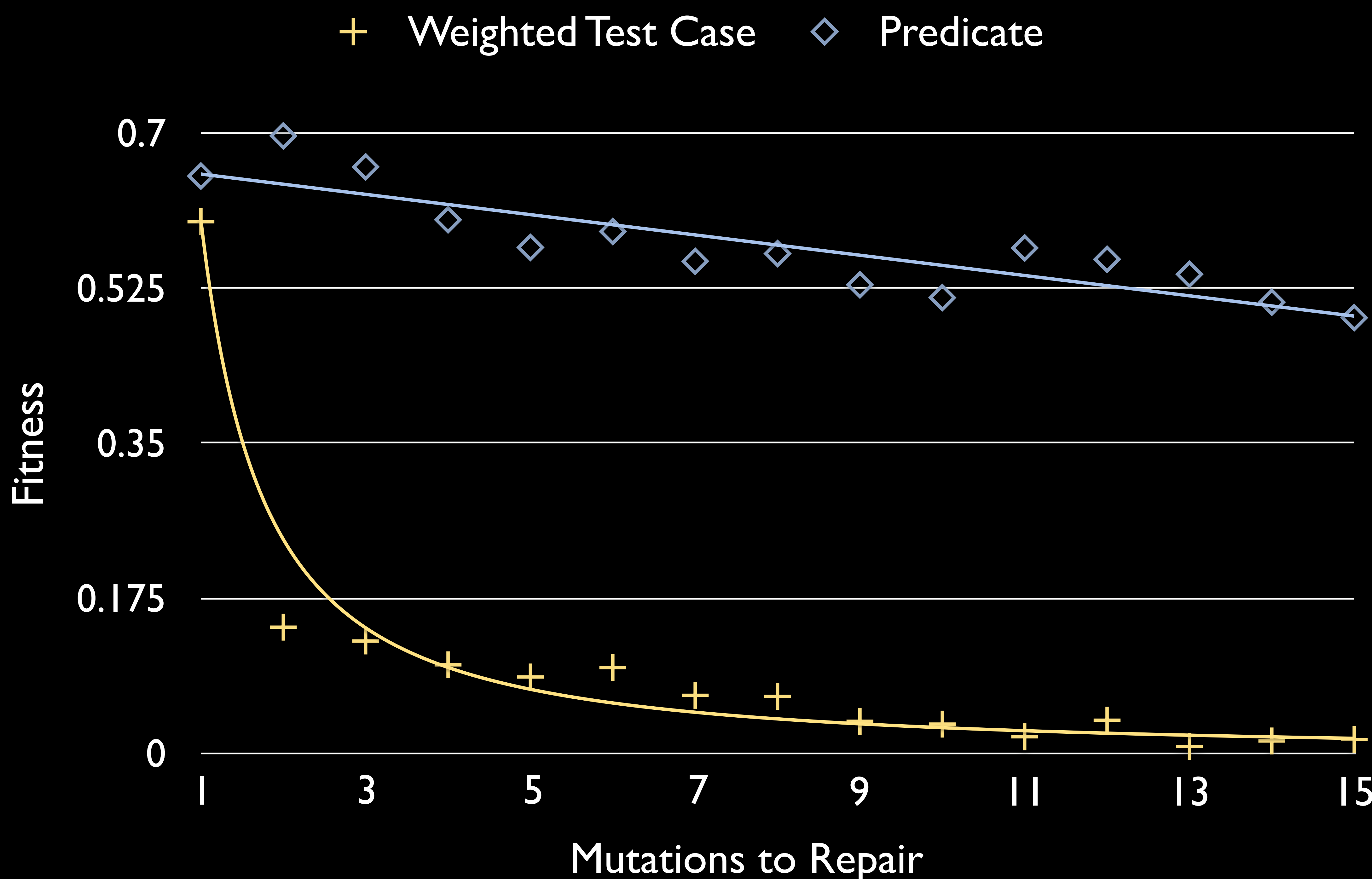
From Predicates to Fitness

- All possible combinations of predicates
- Linear regression identifies a combination of these features predictive of oracle fitness
- Oracle fitness: ideal fitness value that approximates the distance between an individual and the solution
- Hypothesis:
    - Dynamic predicates will enhance the fitness function
    - Evaluate with Fitness Distance Correlation

Predicate-based FDC

- Fitness Distance Correlation: the correlation between a fitness function and the ideal function
- Goal: Design a FF that is "GA-easy" and precise
- Experiment
  - Comparing test-case-only fitness function against new function augmented with predicates
  - Test case based FF from prior work had 0.04 FDC; a "difficult" GA search
  - With predicates, an FDC of 0.63

Nullhttpd Evolution



Findings

- Predicate based function decreases more smoothly than test case based function
- More precise than is possible with any linear combination of test cases
- A more consistent signal evaluates intermediate program variants with greater accuracy
- Currently experimenting with real repairs

Conclusion

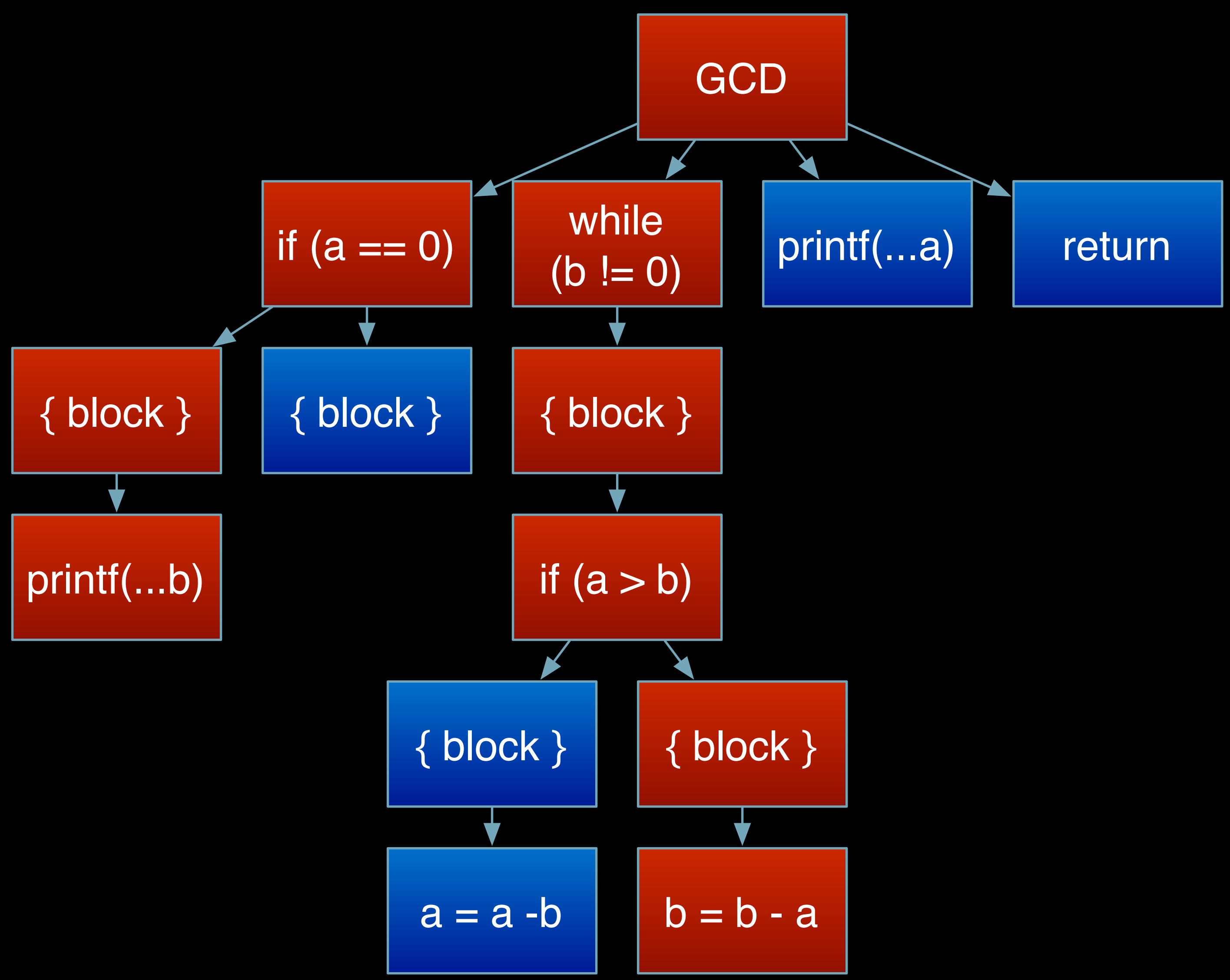
- Two fitness function enhancements
- Efficiency: Test suite sampling
  - Scale to more realistic systems
  - Repaired 10 programs with 206 test cases, an order of magnitude more than previous work
- Precision: Dynamic predicates
  - Significant improvement in FDC (0.04 to 0.63)
  - More smoothly evaluates intermediate variants

Questions?

- Research supported by the *National Science Foundation*, the *Air Force Office of Scientific Research*, and *Microsoft Research*

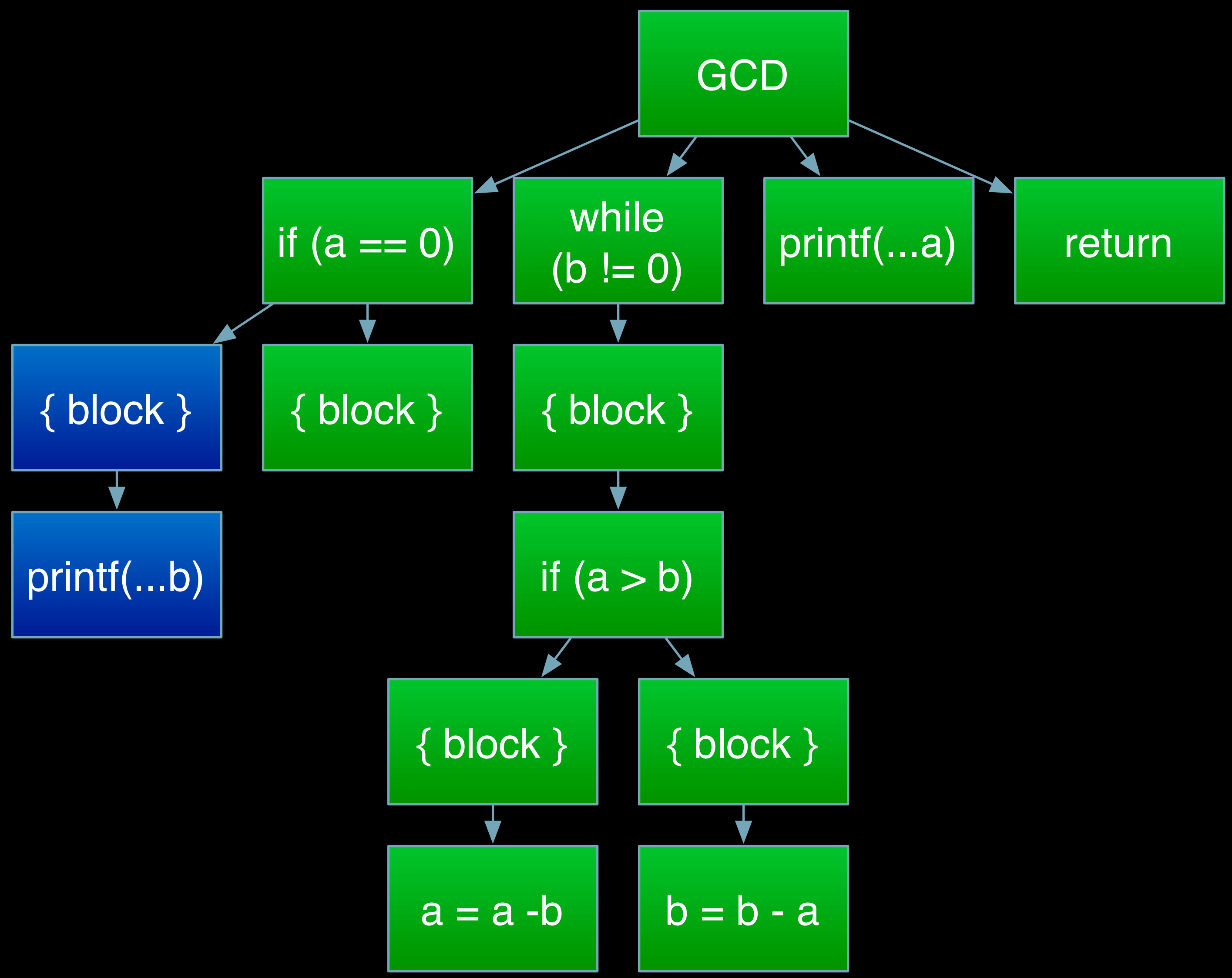
Bonus Slides

Weighted Path



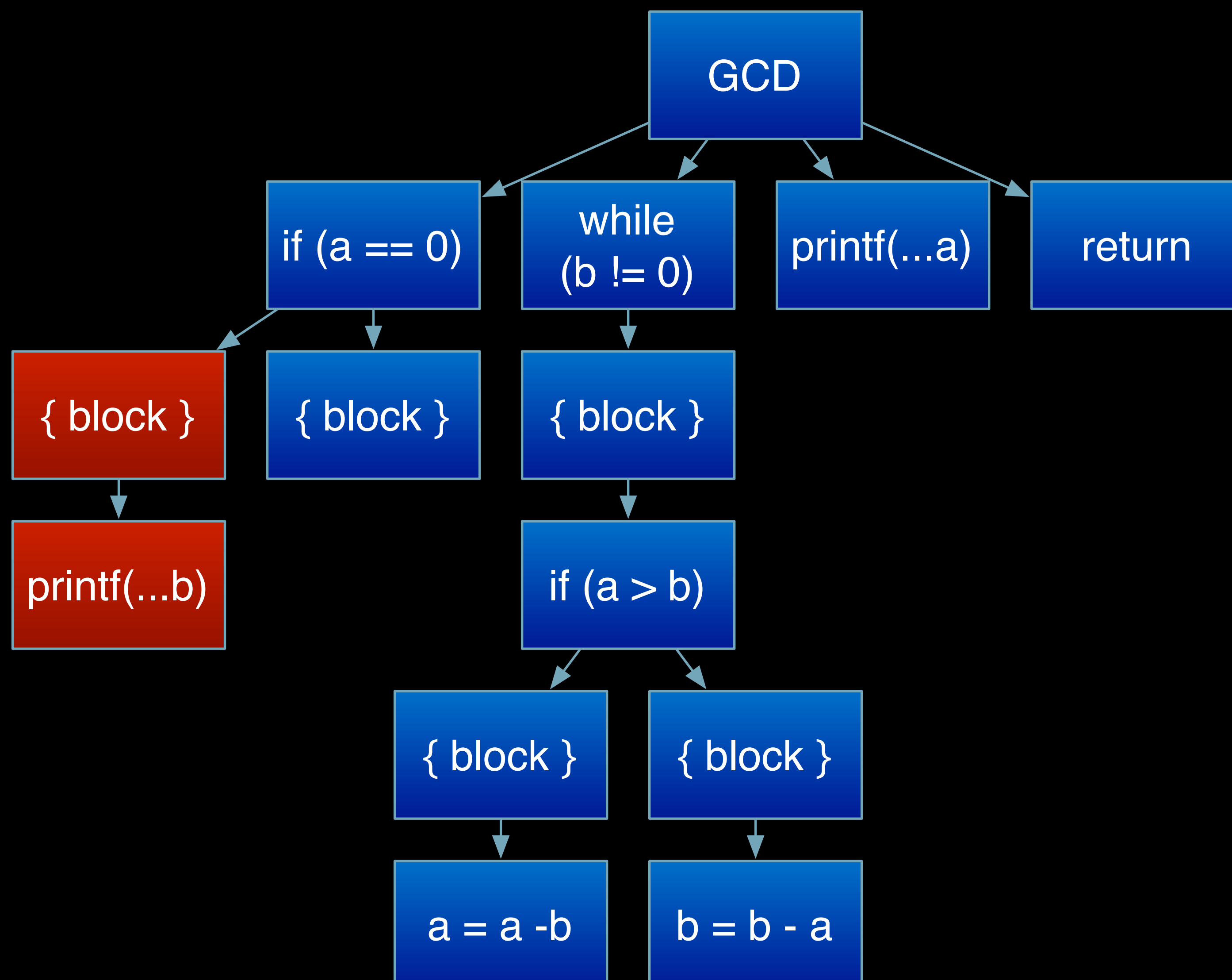
Negative Test Case

Weighted Path



Positive Test Case

Weighted Path



Final Path

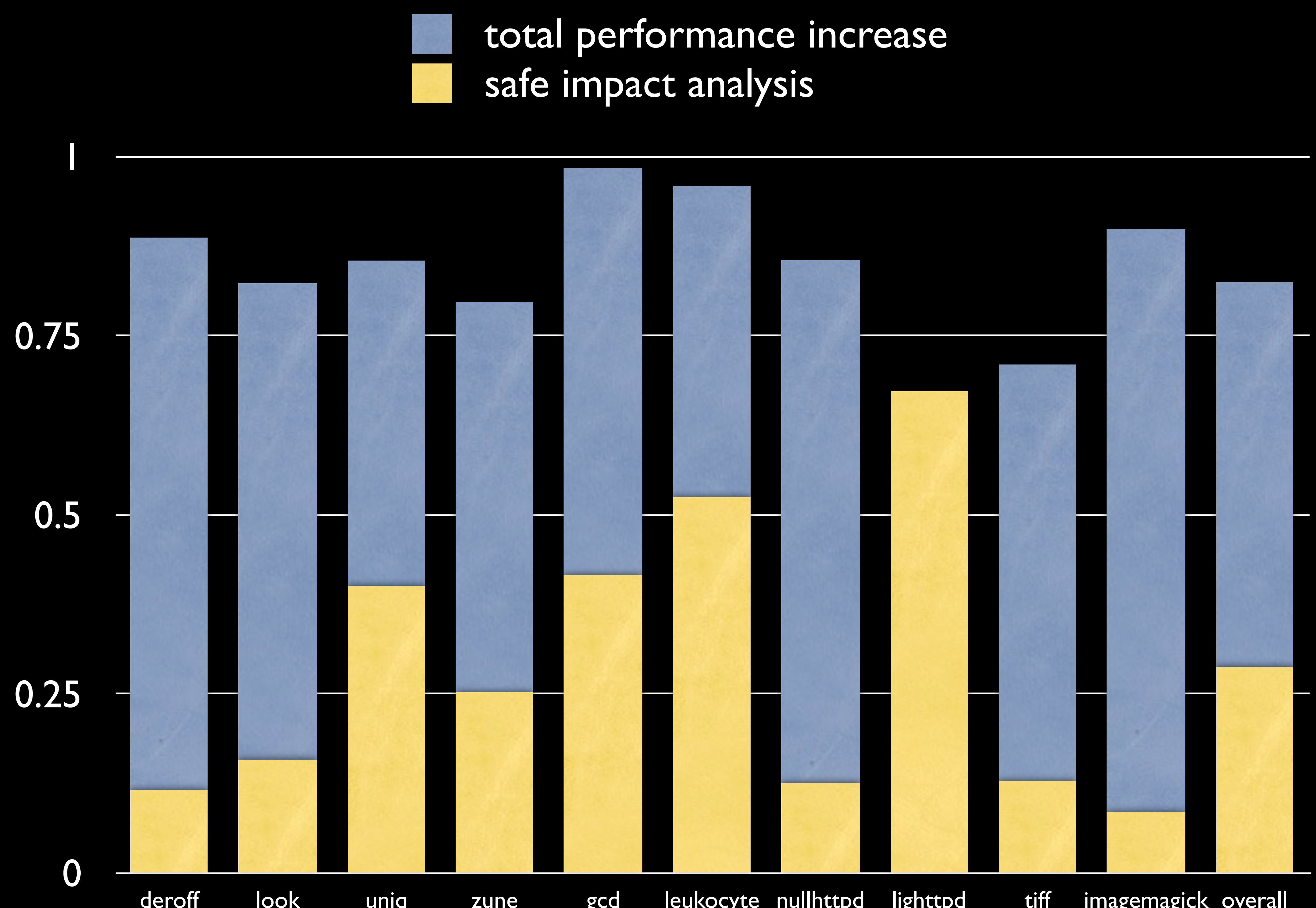
Reduction Example

- Regression Suite
  - R = {T1, T2, T3, T4, T5, T6}
- Subset Selected
  - S = {T1, T4, T6}
- Evaluate fitness on subset
  - Pass T1 and T6
  - Positive Fitness = 2
  - If all tests were passed, then evaluate fitness on R

Change Impact Analysis

- Definition: Determine which tests could possibly be affected by a source code change
- Hypothesis
  - Performance benefits from sampling arise from a high parent/offspring fitness correlation
- Compare against ideal safe impact analysis
  - Measure the percentage of an offspring's test case results shared with its parent

Experimental Results



Findings

- Optimal safe impact analysis reduces atce/r by 29%
- Significantly less than observed efficiency gains
  - 29% vs 81%
- Independence of test cases?
  - No significant difference between high and low overlap test suites

Dynamic Predicates Details

- How much preprocessing?
  - Once per program, creating a baseline set
  - Can be reused across runs
- How do we get them?
  - Instrumentation w/ CIL
- How kinds do we use?
  - How many: 2759
  - Types: branches, return values, scalar pairs
  - Sets: increase, context, and universal