Time-Aware Test Suite Prioritization

Kristen R. Walcott, Mary Lou Soffa
University of Virginia

Gregory M. Kapfhammer, Robert S. Roos
Allegheny College

International Symposium on Software Testing and Analysis

Portland, Maine July 17-20, 2006

Regression Testing

- Software is constantly modified
  - Bug fixes
  - Addition of functionality
- After making changes, test using regression test suite
  - Provides confidence in correct modifications
  - Detects new faults
- High cost of regression testing
  - More modifications › larger test suite
  - May execute for days, weeks, or months
  - Testing costs are very high

Computer Science at the UNIVERSITY of VIRGINIA

Reducing the Cost

- Cost-saving techniques
  - Selection: Use a subset of the test cases
  - Prioritization: Reorder the test cases
- Prioritization methods
  - Initial ordering
  - Reverse ordering
  - Random ordering
  - Based on fault detection ability

Ordering Tests with Fault Detection

- Idea: First run the test cases that will find faults first
- Complications:
  - Different tests may find the same fault
  - Do not know which tests will find faults
- Use coverage to estimate fault finding ability

Prioritization Example

Prioritized Test Suite (with some fault information)

T2	T1	T4	T5	T6	T3
1 fault	7 faults	3 faults	3 faults	3 faults	2 faults
1 min.	9 min.	4 min.	4 min.	4 min.	3 min.

Faults found / minute

1.0	0.778	0.75	0.75	0.75	0.667

- Retesting generally has a time budget
- Is this prioritization best when the time budget is considered?

Contribution: A test prioritization technique that intelligently incorporates a time budget

Fault Aware Prioritization

FAULTS/ TEST CASE	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
T1	X	X		X	X	X	X	X
T2	X							
T3	X				X			
T4		X	X				X	
T5				X		X		X
T6		X		X		X		

TESTING GOAL: Find as many faults as soon as possible

Computer Science at the University of Virginia

Time Budget: 12 minutes

T1	f_1	f_2		f_4	f_5	f_6	f_7	f_8
T2	f_1							
T3	f_1				f_5			
T4		f_2	f_3				f_7	
T5				f_4		f_6		f_8
T6		f_2		f_4		f_6		

Fault-based Prioritization

T1	T4	T5	T6	T3	T2
7 faults	3 faults	3 faults	3 faults	2 faults	1 fault
9 min.	4 min.	4 min.	4 min.	3 min.	1 min.

Finds 7 unique faults in 9 minutes

Time Budget: 12 minutes

T1	f_1	f_2		f_4	f_5	f_6	f_7	f_8
T2	f_1							
T3	f_1				f_5			
T4		f_2	f_3				f_7	
T5				f_4		f_6		f_8
T6		f_2		f_4		f_6		

Naïve Time-based Prioritization

T2	T3	T4	T5	T6	T1
1 fault	2 faults	3 faults	3 faults	3 faults	7 faults
1 min.	3 min.	4 min.	4 min.	4 min.	9 min.

Finds 8 unique faults in 12 minutes

Time Budget: 12 minutes

T1	f_1	f_2		f_4	f_5	f_6	f_7	f_8
T2	f_1							
T3	f_1				f_5			
T4		f_2	f_3				f_7	
T5				f_4		f_6		f_8
T6		f_2		f_4		f_6		

Average-based Prioritization

T2	T1	T4	T5	T6	T3
1 fault	7 faults	3 faults	3 faults	3 faults	2 faults
1 min.	9 min.	4 min.	4 min.	4 min.	3 min.

Finds 7 unique faults in 10 minutes

Computer Science at the University of Virginia

Time Budget: 12 minutes

T1	f_1	f_2		f_4	f_5	f_6	f_7	f_8
T2	f_1							
T3	f_1				f_5			
T4		f_2	f_3				f_7	
T5				f_4		f_6		f_8
T6		f_2		f_4		f_6		

Intelligent Time-Aware Prioritization

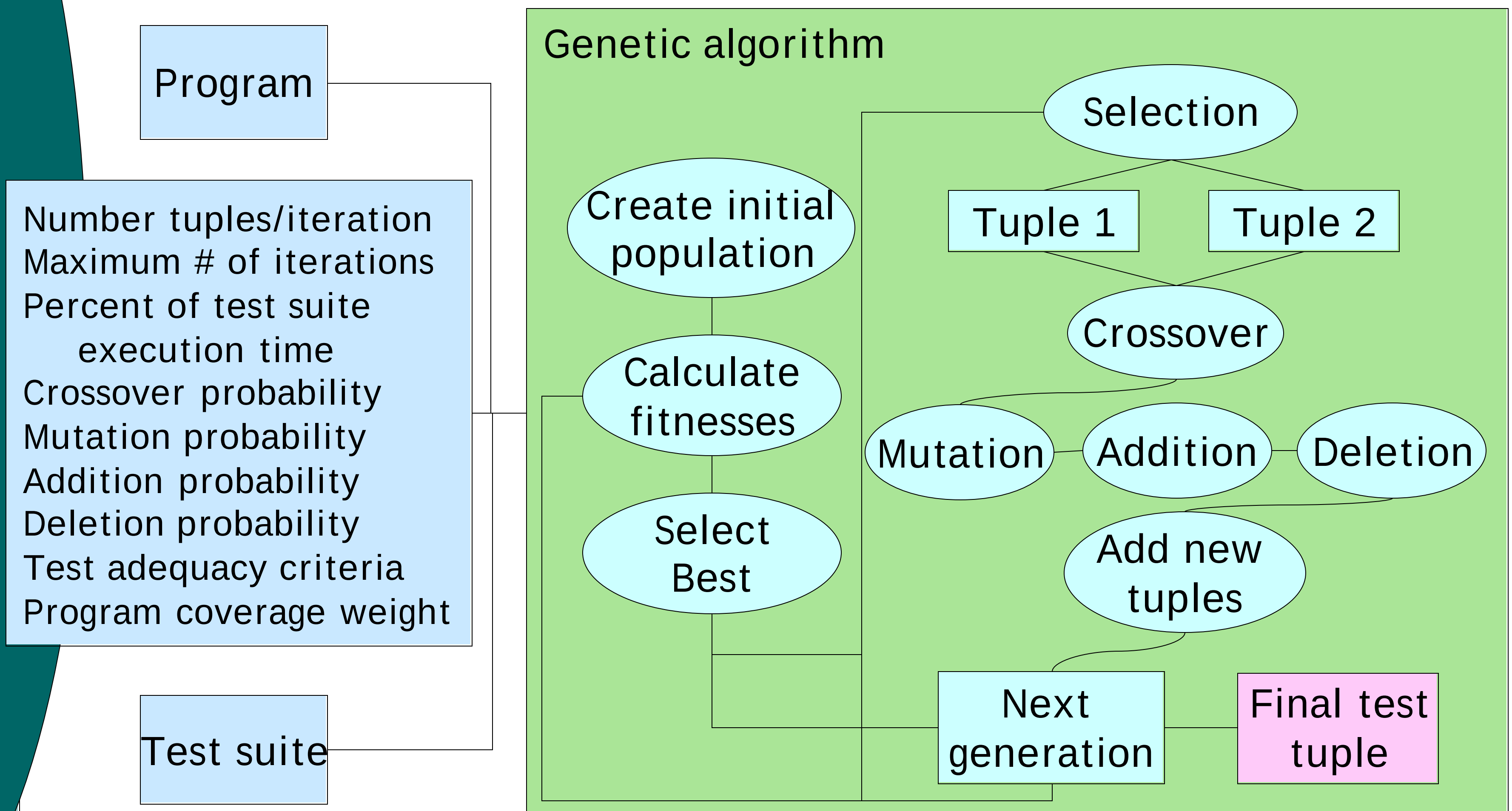
T5 3 faults 4 min.	T4 3 faults 4 min.	T3 2 faults 3 min.	T1 7 faults 9 min.	T2 1 fault 1 min.	T6 3 faults 4 min.

Finds 8 unique faults in 11 minutes

Time-Aware Prioritization

- Time-aware prioritization (TAP) combines:
  - Fault finding ability (overlapping coverage)
  - Test execution time
- Time constrained test suite prioritization problem   0/1 knapsack problem
  - Use genetic algorithm heuristic search technique
  - Genetic algorithm
    - Fitness ideally calculated based on faults
    - A fault cannot be found if code is not covered
    - Fitness function based on test suite and test case code coverage and execution time

Prioritization Infrastructure



Fitness Function

Primary Fitness
Secondary Fitness

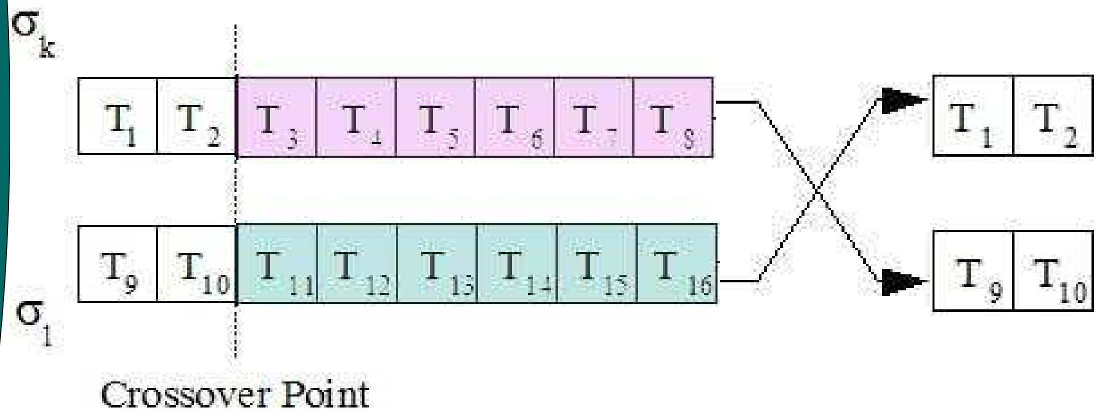
Test Suite 1: 70% coverage ← Preferred!

Test Suite 2: 40% coverage

- Fitness function components
  1. Overall coverage
  2. Cumulative coverage of test tuple
  3. Time required by test tuple
     - If over time budget, receives very low fitness

Creation of New Test Tuples

Crossover



- Vary test tuples using recombination
- If recombination causes duplicate test case execution, replace duplicate test case with one that is unused

Computer Science at the University of Virginia

Creation of New Test Tuples

- Mutation
  - For each test case in tuple
    - Select random number, R
    - If R < mutation probability, replace test case
- Addition- Append random unused test case
- Deletion- Remove random test case

Computer Science at the University of Virginia

Experimentation Goals

- Analyze trends in average percent of faults detected (APFD)
- Determine if time-aware prioritizations outperform selected set of other prioritizations
- Identify time and space overheads

Experiment Design

- GNU/Linux workstations
  - 1.8 GHz Intel Pentium 4
  - 1 GB main memory
- JUnit test cases used for prioritization
- Case study applications
  - Gradebook
  - JDepend
- Faults seeded into applications
  - 25, 50, and 75 percent of 40 errors

Computer Science at the University of Virginia

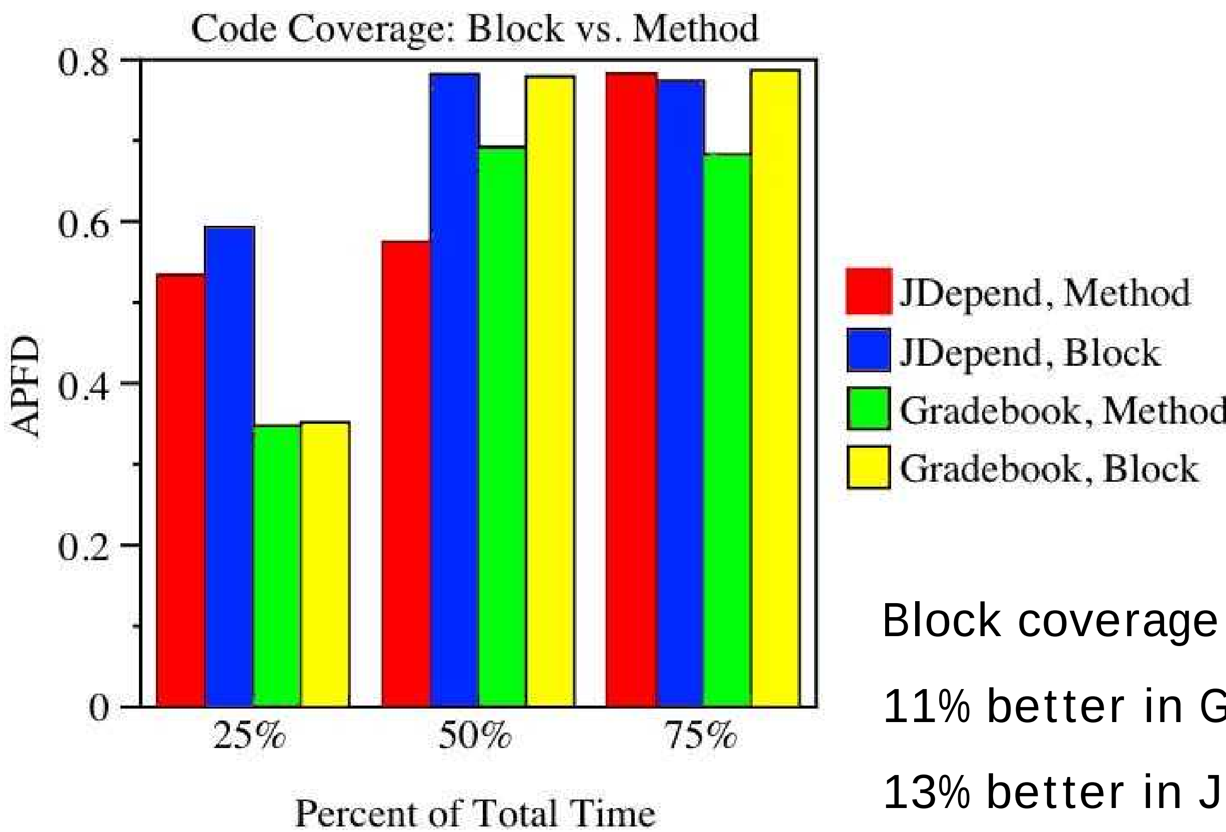
Evaluation Metrics

- Average percent of faults detected (APFD)
  - T = test tuple
  - g = number of faults in program under test
  - n = number of test cases
  - *reveal(i, T)* = position of the first test in T that exposes fault i

$$APFD(T,P) = 1 - \frac{\sum_{i=1}^{g} reveal(i,T)}{ng} + \frac{1}{2n}$$

- Peak memory usage
- User and system time

TAP APFD Values

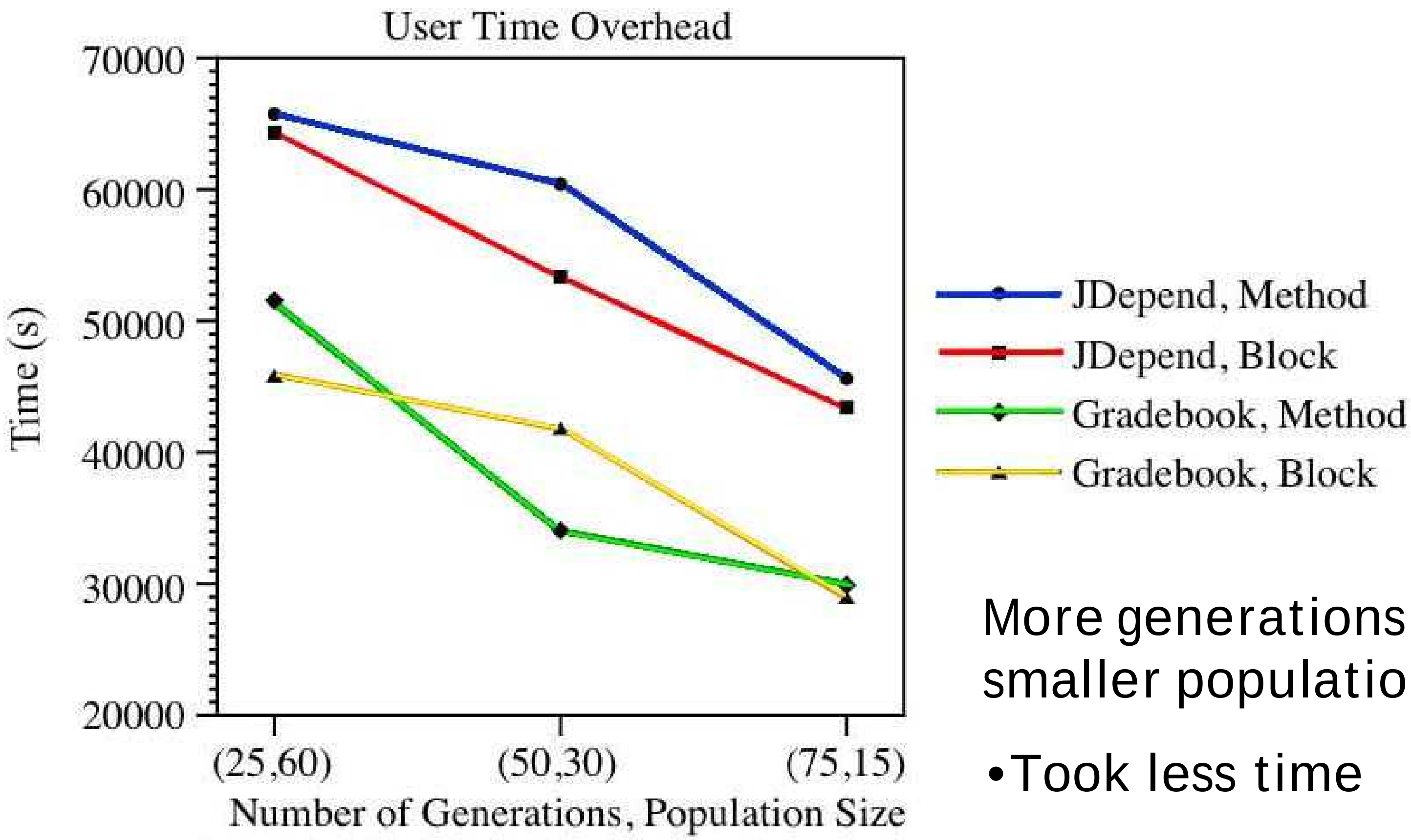


Block coverage preferred:

11% better in Gradebook

13% better in JDepend

TAP Time Overheads

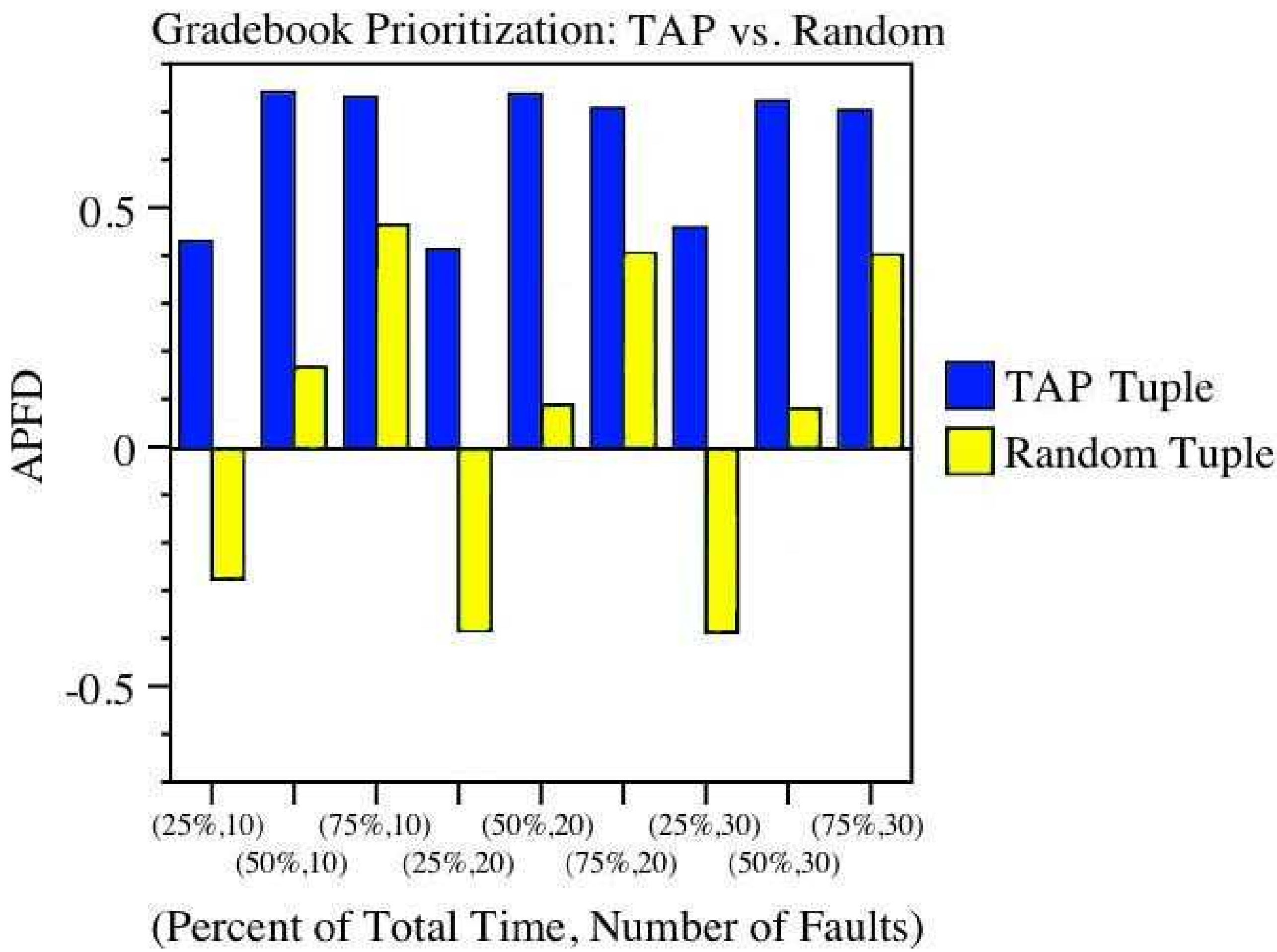


More generations with smaller populations:

- Took less time
- Same quality results

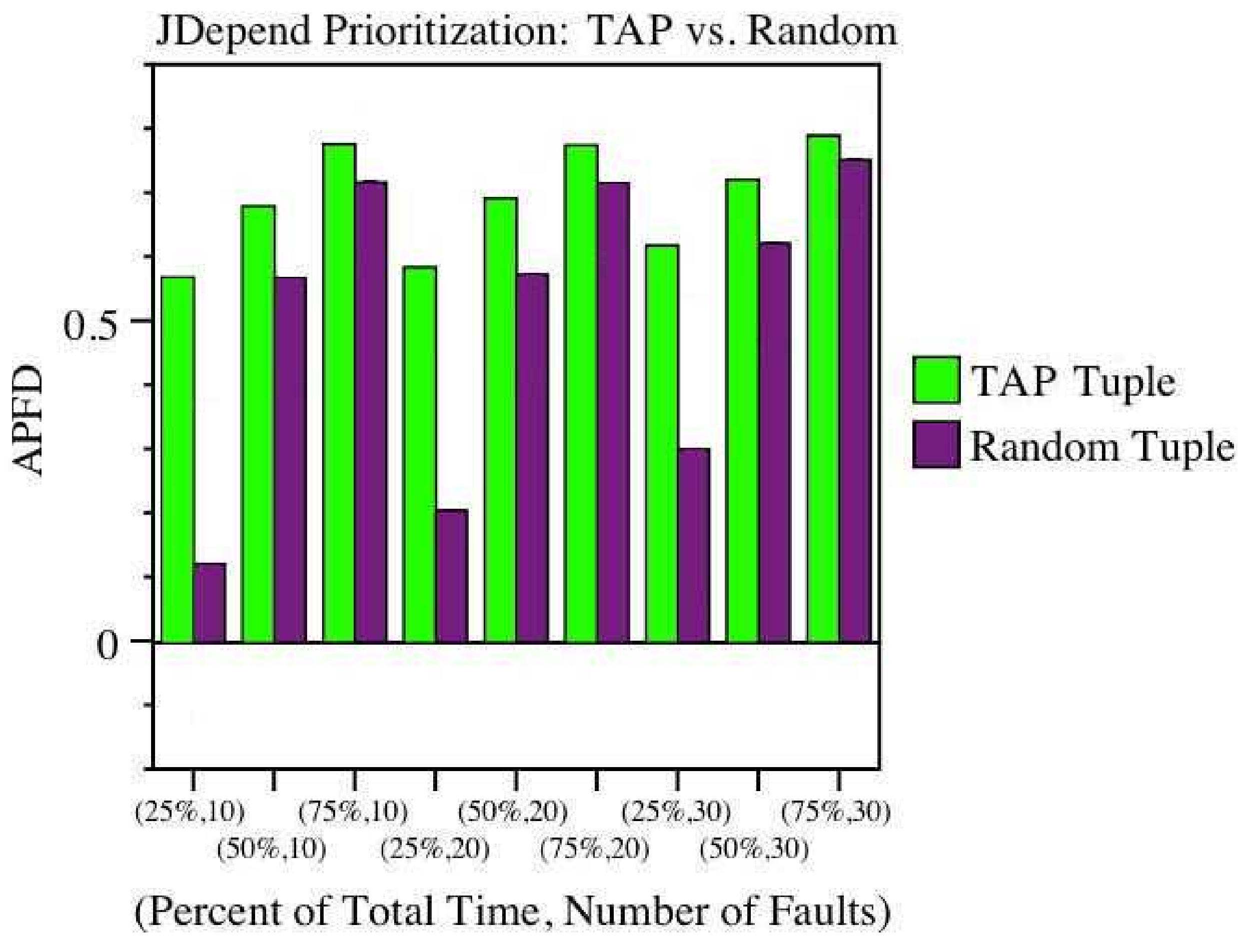
Computer Science at the University of Virginia

Gradebook: Intelligent vs Random



Computer Science at the University of Virginia

JDepend: Intelligent vs. Random



Computer Science at the University of Virginia

Other Prioritizations

- Random prioritizations redistribute fault-revealing test cases
- Other prioritizations
  - Initial ordering
  - Reverse ordering
  - Fault-aware
    - Impossible to implement
    - Good watermark for comparison

Gradebook: Alternative Prioritizations

% total time	# Faults	Initial	Reverse	TAP	Fault aware
0.25	10	-0.6	-0.2	0.43	0.7
0.25	20	-0.9	-0.2	0.41	0.7
0.25	30	-0.9	-0.0	0.46	0.5
0.50	10	-0.04	0.1	0.74	0.9
0.50	20	-0.2	0.2	0.74	0.9
0.50	30	-0.3	0.3	0.72	0.8
0.75	10	0.3	0.5	0.73	0.9
0.75	20	0.1	0.4	0.71	0.9
0.75	30	0.04	0.5	0.70	0.9

- Time-aware prioritization up to 120% better than other prioritizations

Computer Science at the University of Virginia

Conclusions and Future Work

- Analyzes a test prioritization technique that accounts for a testing time budget
- Time intelligent prioritization had up to 120% APFD improvement over other techniques

- Future Work
  - Make fitness calculation faster
  - Distribute fitness function calculation
  - Exploit test execution histories
  - Create termination condition based on prior prioritizations
  - Analyze other search heuristics

Computer Science at the UNIVERSITY of VIRGINIA

Thank you!

Time-Aware Prioritization (TAP) Research:

- http://www.cs.virginia.edu/~krw7c/TimeAwarePrioritization.htm

Computer Science at the University of Virginia