Using Synthetic Coverage Information to Evaluate Test Suite Prioritizers

Gregory M. Kapfhammer†

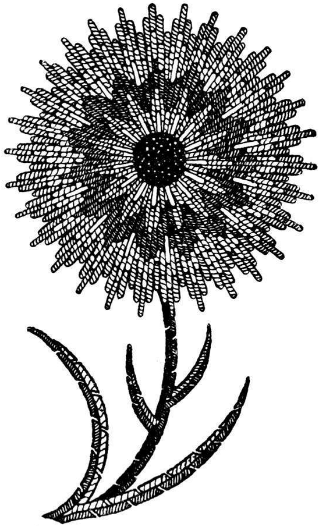
Department of Computer Science
Allegheny College, Pennsylvania, USA
http://www.cs.allegheny.edu/~gkapfham/

Chennai Mathematical Institute, February 2008

† In Conjunction with Mary Lou Soffa, Kristen Walcott (UVa/CS)
Suvarshi Bhadra, Joshua Geiger, Adam Smith, Gavilan Steinman, Yuting Zhang (Allegheny/CS)

Featuring images from *Embroidery and Tapestry Weaving*, Grace Christie (Project Gutenberg)

Presentation Outline



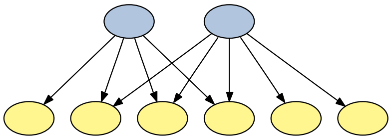
The Challenge of Software Testing

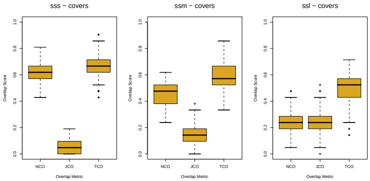
I shall not deny that the construction of these testing programs has been a major intellectual effort: to convince oneself that one has not overlooked “a relevant state” and to convince oneself that the testing programs generate them all is no simple matter. The encouraging thing is that (as far as we know!) it could be done.

Edsger W. Dijkstra, *Communications of the ACM*, 1968

Additional Challenge: empirically evaluating the efficiency and effectiveness of software testing techniques

Important Contributions



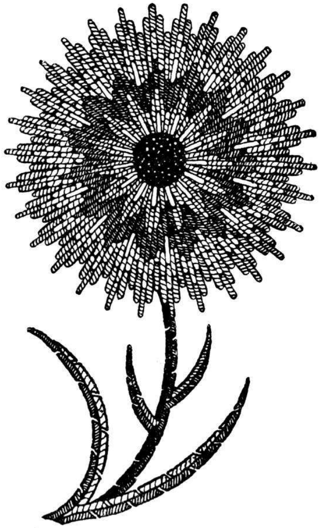


Synthetic Coverage Generators

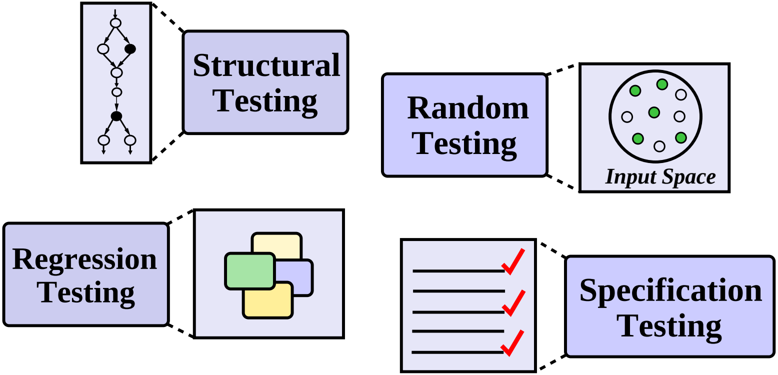
Detailed Experimental Results

A **comprehensive framework** that supports the empirical evaluation of regression test suite prioritizers

Presentation Outline

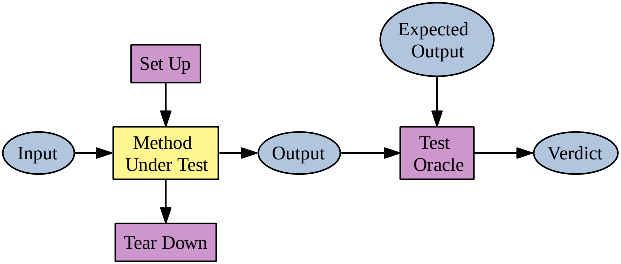


Approaches to Software Testing



Testing **isolates defects** and establishes a **confidence in the correctness** of a software application

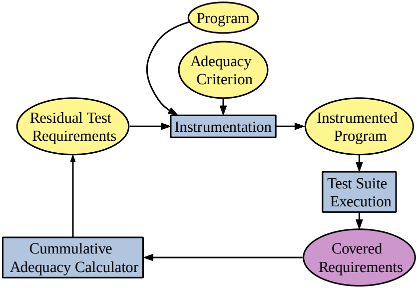
What is a Test Case?



Overview

- Test suite executor runs each test case **independently**
- Each test invokes a method within the program and then compares the **actual** and **expected** output values

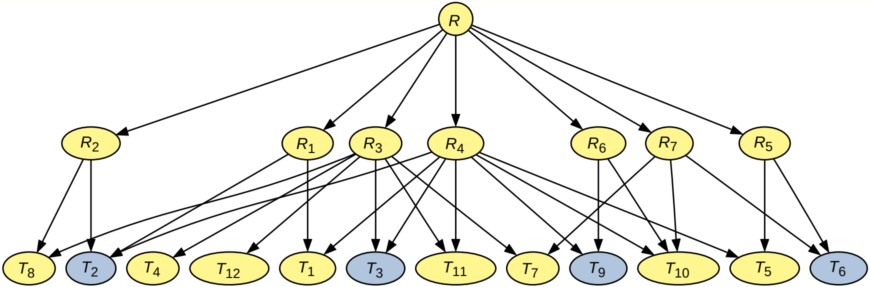
Test Coverage Monitoring



Overview

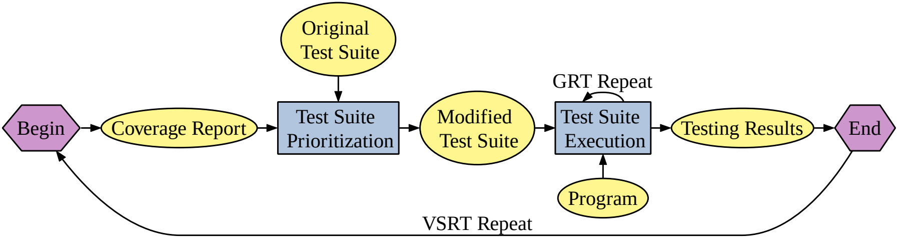
- Structural **adequacy criteria** focus on the coverage of nodes, edges, paths, and definition-use associations
- Instrumentation **probes** track the coverage of test requirements

Finding the Overlap in Coverage



- $R_j \rightarrow T_i$ means that requirement R_j is **covered by** test T_i
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements
- Include the **remaining** tests so that they can **redundantly** cover the requirements

Regression Test Suite Prioritization

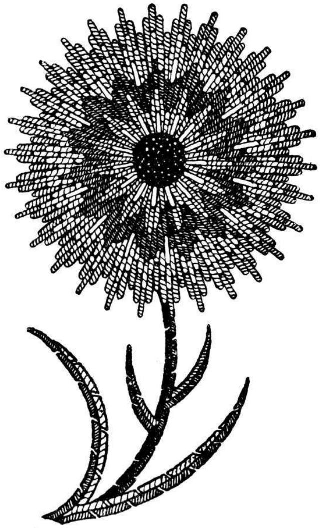


Overview

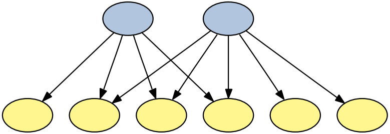
- Prioritization **re-orders** the tests so that they cover the requirements more effectively
- Researchers and practitioners need to determine whether the **prioritized** test suite is better than the **original** ordering

Presentation Outline

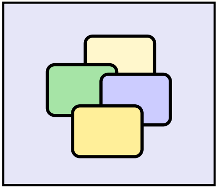
1. Challenges and Solutions
2. Regression Testing Techniques
3. **Conducting Empirical Studies**
4. Generating Synthetic Coverage
5. Empirical Evaluation
6. Future Work
7. Conclusions



Using Real World Applications



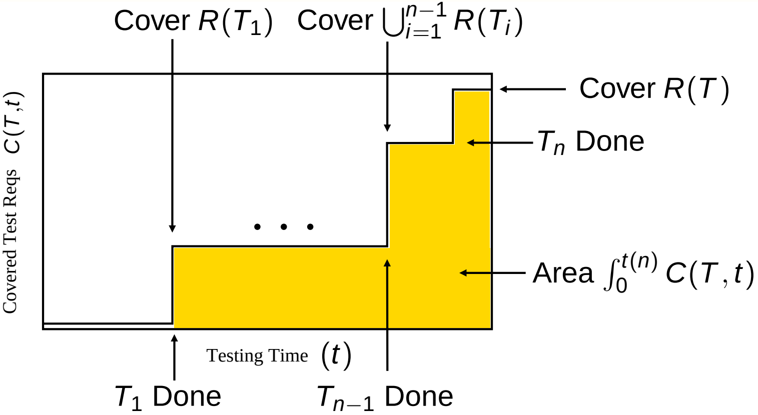
Regression Test Suite



Program Under Test

It is difficult to **systematically** study the efficiency and effectiveness trade-offs because coverage overlap **varies**

Coverage Effectiveness Metric



- Prioritize to **increase** the CE of a test suite $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0, 1]$

Characterizing a Test Suite

Test Information

Test Case	Cost (sec)	Requirements				
		R_1	R_2	R_3	R_4	R_5
T_1	5	✓	✓			
T_2	10	✓	✓	✓		✓
T_3	4	✓			✓	✓

Total Testing Time = 19 seconds

Formulating the Metrics

CE considers the **execution time** of each test while CE_u assumes that all test cases execute for a **unit cost**

Coverage Effectiveness Values

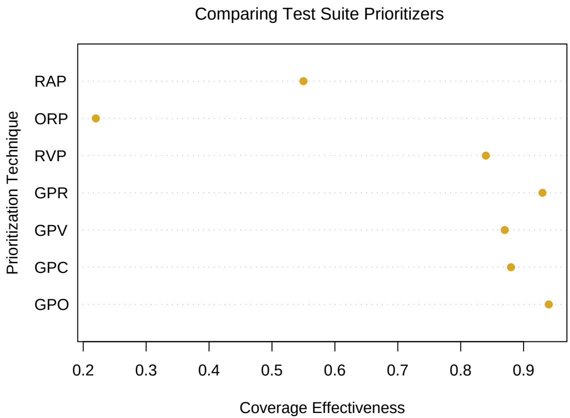
Calculating CE and CE_u

Ordering	CE	$\mathbf{CE_u}$
T_1 T_2 T_3	.3789	.4
T_1 T_3 T_2	.5053	.4
T_2 T_1 T_3	.3789	.5333
T_2 T_3 T_1	.4316	.6
T_3 T_1 T_2	.5789	.4557
T_3 T_2 T_1	.5789	.5333

Observations

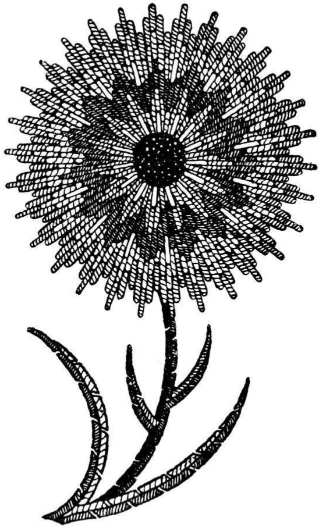
- Including test case costs does impact the CE metric
- Depending upon the characteristics of the test suite, we may see $CE = CE_u$, $CE > CE_u$, or $CE < CE_u$

Comparing Prioritization Techniques



Does this result **generalize** to other applications?

Presentation Outline



Test Suites and Requirements

Regression Test Suite

$$T = \langle T_1, \ldots, T_n \rangle$$
$$T_i \in T$$

Test Requirements

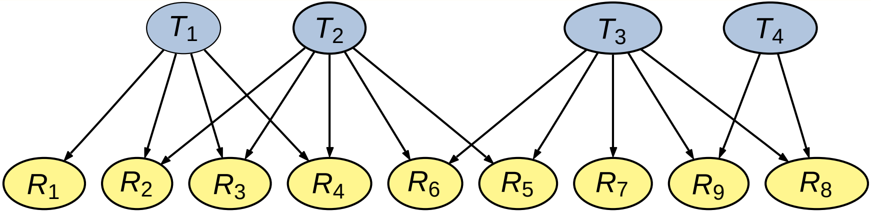
$$R = \{R_1, \ldots, R_m\}$$
$$R_j \in R$$

covers(i) denotes the set of **requirements** that T_i covers

coveredby(j) denotes the set of **test cases** that cover R_j

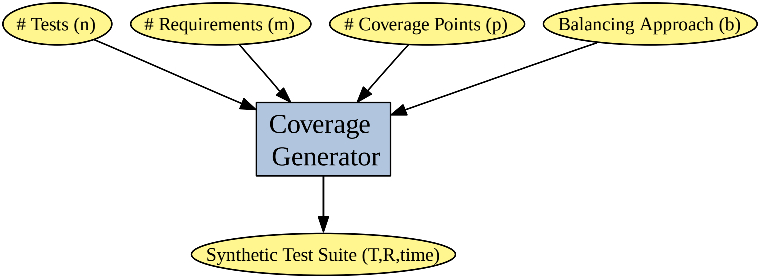
Goal: **automatically** generate a **synthetic** regression test suite T that covers the requirements in R

Coverage Overlap Metrics



$$\begin{aligned}
NCO(i,k) &= (R \setminus covers(i)) \cap (R \setminus covers(k)) \\
NCO(1,2) &= \{R_7, R_8, R_9\} \\
JCO(i,k) &= covers(i) \cap covers(k) \\
JCO(1,2) &= \{R_2, R_3, R_4\} \\
TCO(i,k) &= NCO(i,k) \cup JCO(i,k) \\
TCO(1,2) &= \{R_2, R_3, R_4, R_7, R_8, R_9\}
\end{aligned}$$

Standard Coverage Generation



Generation Procedure

- Guarantee that **each requirement** is covered by a test case and that **all tests** cover at least one requirement
- **Balance** the coverage information according to the **cardinality** of either the $covers(i)$ or the $coveredby(j)$ sets

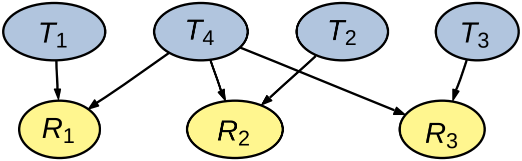
Configuring the Standard Generator

Number	Small	Medium	Large
Tests (n)	10	50	100
Requirements (m)	$2 \times n$	$5 \times n$	$10 \times n$
Coverage Points (p)	$(n \times m)/5$	$(n \times m)/3$	$(n \times m)/2$

Generating Coverage

- Configuration **sss** generates 10 tests, 20 requirements, and 40 coverage points
- Configuration **lll** generates 100 tests, 1000 requirements, and $50,000$ coverage points
- For all of the above configurations, the **generation procedure** consumes less than **one second** of execution time

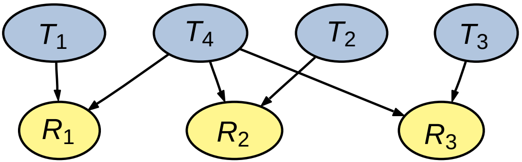
"Greedy Fooling" Coverage Generation



Generation Procedure

- The **greedy** test prioritizer iteratively selects test cases according to the **(coverage / cost)** ratio
- **Goal**: generate coverage and timing information that will **fool** the greedy technique into creating $T' = \langle T_n, \ldots, T_1 \rangle$ even though $CE(T') < CE(T)$ for $T = \langle T_1, \ldots, T_n \rangle$
- **Inspiration**: Vazirani's construction of a **tight example** for the greedy **minimal set cover** algorithm

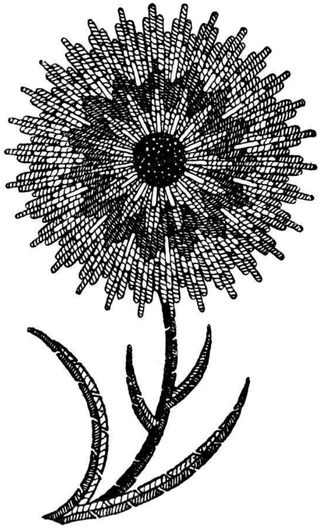
Constructing "Greedy Fooling" Test Suites



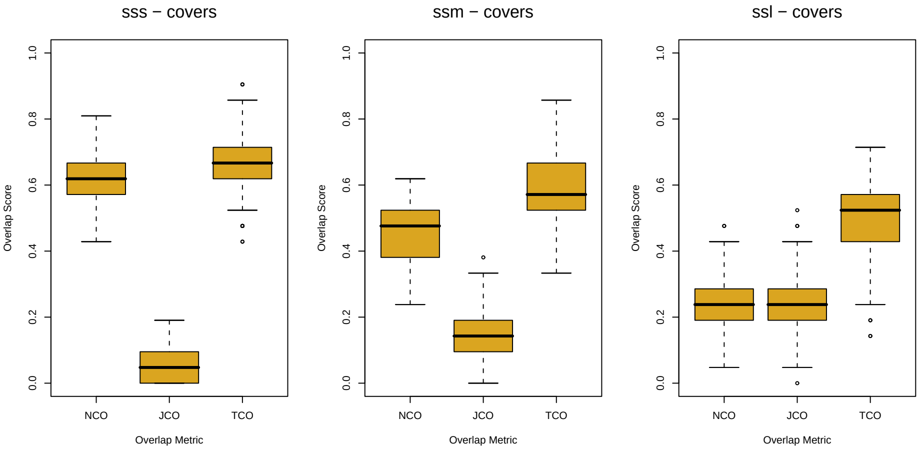
- **Approach**: use one dimensional **optimization** (e.g., golden section search and successive parabolic interpolation) to pick a value for $cost(T_n)$
- **Construction**: set $cost(T_1) = cost(T_2) = cost(T_3) = 1$ and then determine the bounds for $cost(T_4) \in [C_{min}, C_{max}]$
- **Example**: $cost(T_4) \in [2.138803, 2.472136]$ so that
  $CE_{min}(T') = .5838004 \quad CE_{min}(T) = .6108033$
  $CE_{max}(T') = .5482172 \quad CE_{max}(T) = .6345125$

Presentation Outline

1. Challenges and Solutions
2. Regression Testing Techniques
3. Conducting Empirical Studies
4. Generating Synthetic Coverage
5. **Empirical Evaluation**
6. Future Work
7. Conclusions

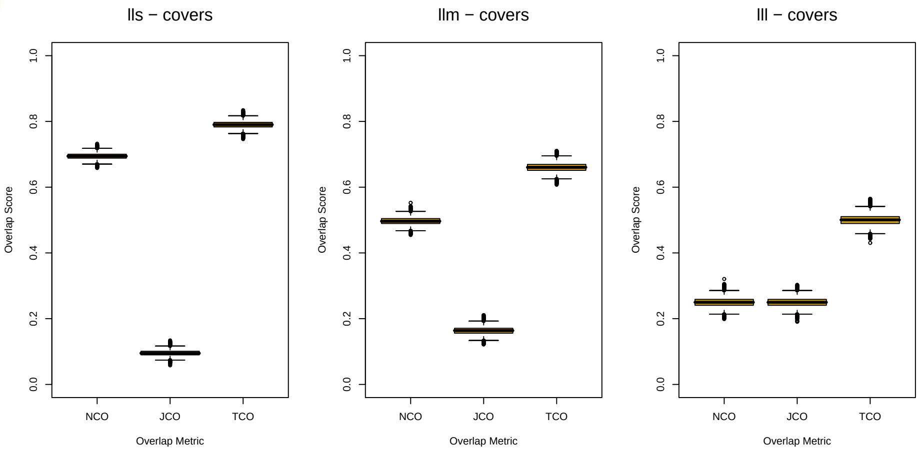


Overlap Metrics - Small Test Suite



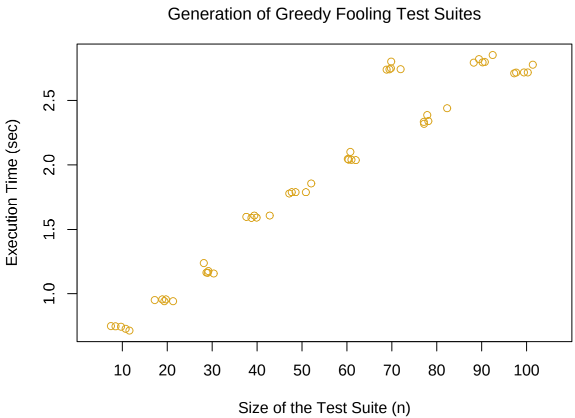
Increasing the value of *p* **changes** the coverage overlap metrics

Overlap Metrics - Large Test Suite



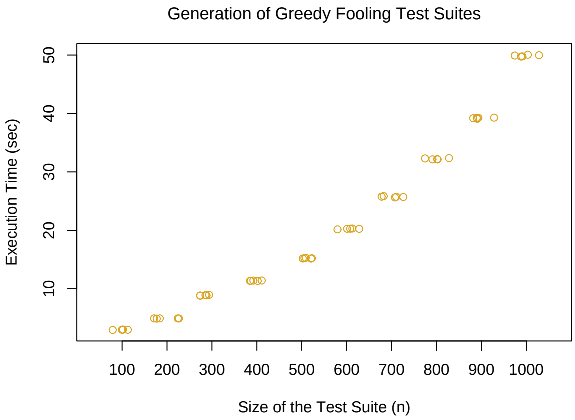
Increasing test suite size **tightens** the coverage overlap metrics

Greedy Fooling Time - Small Test Suite



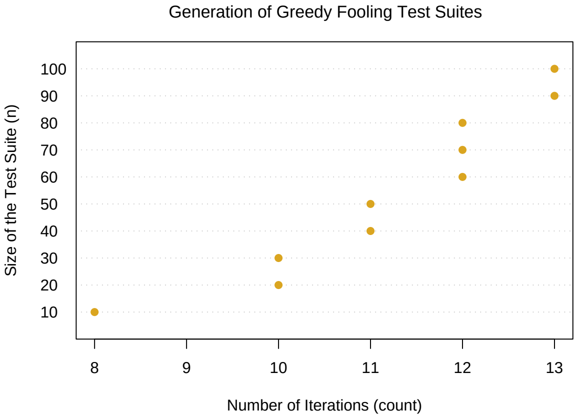
The generation of a small test suite takes less than 3 seconds

Greedy Fooling Time - Large Test Suite



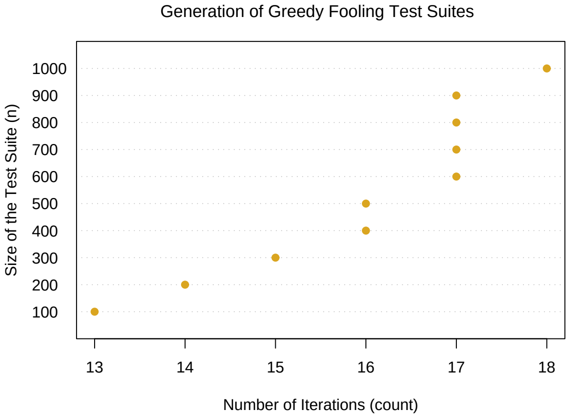
The generation of a large test suite takes up to 50 seconds

Greedy Fooling Iterations - Small Test Suite



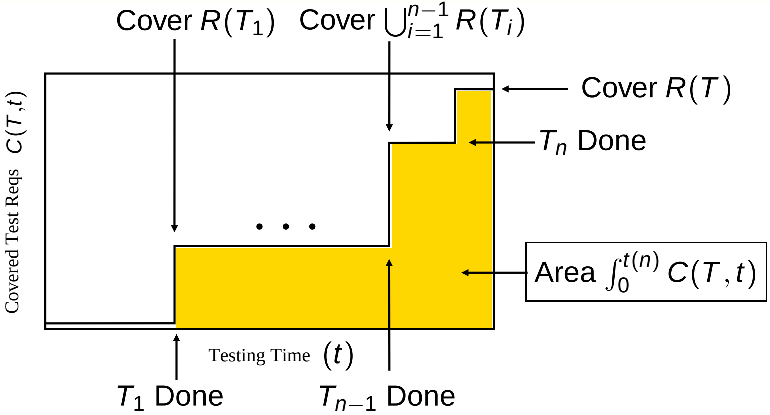
Finding a bound for $cost(T_n)$ requires few iterations of the optimizer

Greedy Fooling Iterations - Large Test Suite



Increasing the value of *n* does not markedly increase the iteration count

Cost of Coverage Generation

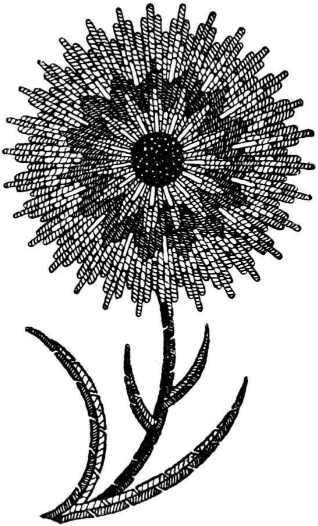


The cost of generation is **dominated** by **numerical integration**'s cost

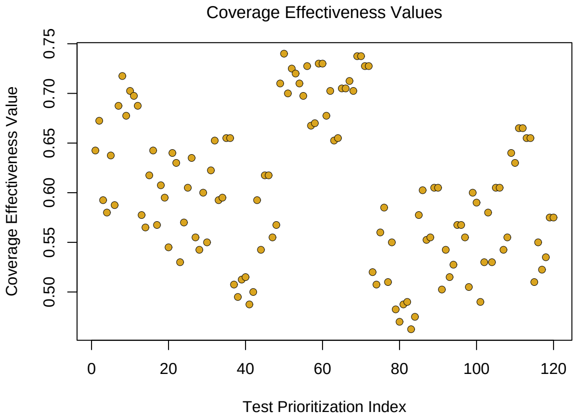
Fooling the Greedy Prioritizer

n	C_{min}	C_{max}	$CE_{min}(T')$	$CE_{max}(T')$	$CE_{min}(T)$	$CE_{max}(T)$
10	5.2786	8.541	0.63031	0.51308	0.64983	0.71519
20	10.1320	18.885	0.65222	0.50150	0.65670	0.73680
30	15.1970	28.967	0.65616	0.50000	0.66076	0.74138
40	20.2630	38.622	0.65809	0.50243	0.66256	0.74239
50	25.3290	48.936	0.65922	0.50000	0.66354	0.74490
60	30.0610	58.723	0.66246	0.50117	0.66320	0.74514
70	35.1090	68.510	0.66276	0.50178	0.66377	0.74551
80	40.1240	78.948	0.66318	0.50000	0.66429	0.74684
90	45.1400	88.816	0.66347	0.50000	0.66448	0.74693
100	50.1550	98.684	0.66374	0.50000	0.66460	0.74707

Presentation Outline

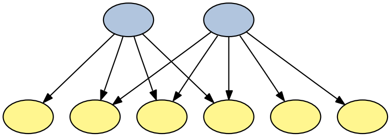


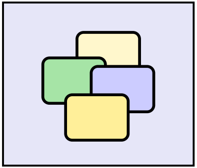
Search-Based Test Suite Prioritization



Use **heuristic search** (HC, SANN, GA) to prioritize the test suite

Detailed Empirical Evaluations



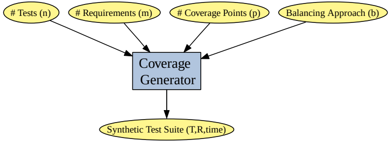


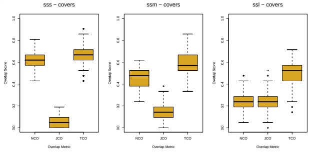
Synthetic Test Suites

Real World Programs

Systematically study the efficiency and effectiveness trade-offs with **synthetic** coverage and then conduct further experimental studies with **real world** applications

Concluding Remarks





Synthetic Coverage Generators

Detailed Experimental Results

A **comprehensive framework** that furnishes a **new perspective** on the empirical evaluation of regression test suite prioritizers

http://www.cs.allegheny.edu/~gkapfham/