

Set Covers, Knapsacks, and Regression Testing Techniques

Gregory M. Kapfhammer[†]

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

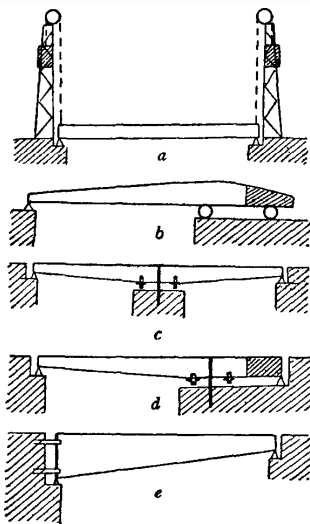
Madras Christian College, February 2008

[†] 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 the WikiMedia Commons

Presentation Outline

- 1 Testing Challenges
- 2 Reduction and Prioritization
- 3 Time-Aware Prioritization
- 4 Future Work
- 5 Conclusions



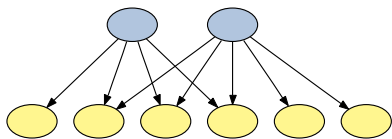
The Challenge of Software Testing

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.

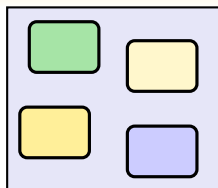
Edsger W. Dijkstra, *Communications of the ACM*, 1968

Additional Challenge: understanding the **fundamental** difficulties associated with **practical** testing techniques

NP-Complete Problems



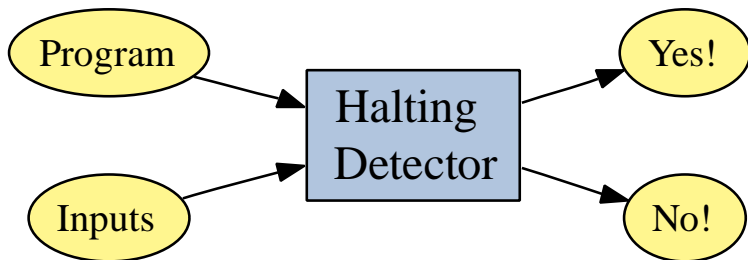
Minimal Set Cover



0/1 Knapsack

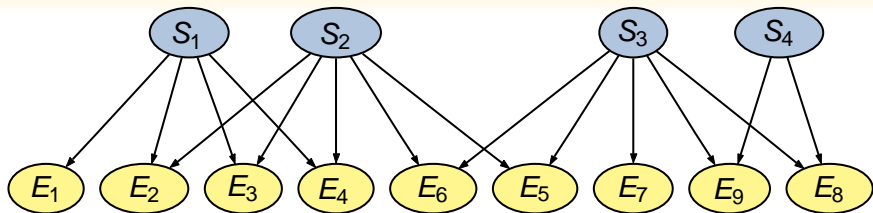
Question: what are the connections between the **theory** and **practice** of computer science?

Halting Problem



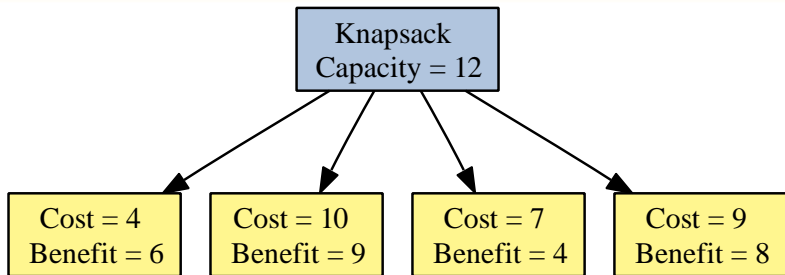
- **Question:** What approach can we take in order to completely implement the halting detector?
- You can assume **any** existing hardware platform (e.g., fast multi-core processor) or software application (e.g., compiler).

Minimal Set Cover Problem



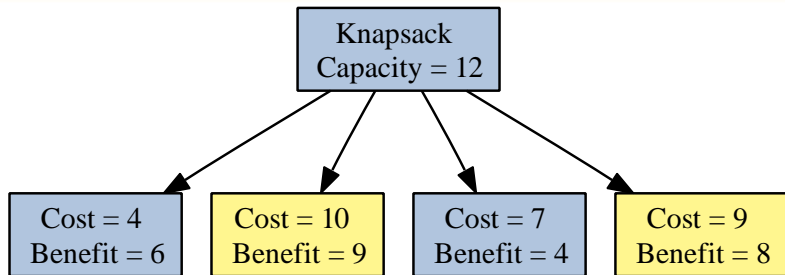
- Sets contain elements (e.g., $S_2 \rightarrow E_4$ means that set S_2 contains the element E_4)
- **Question:** Can you find a **subset** of the sets that will contain all of the elements?
- This problem is **NP-complete** (see Garey and Johnson) and yet it has many practical applications in software testing

0/1 Knapsack Problem



- **Question:** Can you select items so that you **maximize** the benefit while ensuring that the cost does not **exceed** the capacity?
- This problem is **NP-complete** (see Garey and Johnson) and yet it also has many practical applications in both software and finance

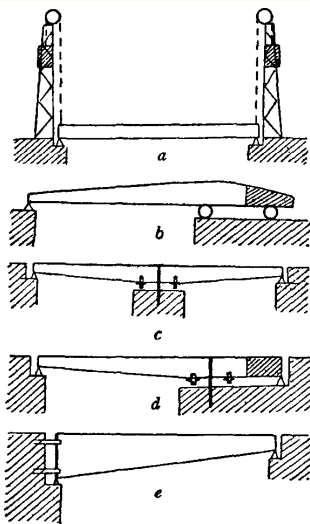
0/1 Knapsack Problem



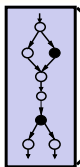
- **Question:** Can you select items so that you **maximize** the benefit while ensuring that the cost does not **exceed** the capacity?
- This problem is **NP-complete** (see Garey and Johnson) and yet it also has many practical applications in both software and finance

Presentation Outline

- 1 Testing Challenges
- 2 Reduction and Prioritization**
- 3 Time-Aware Prioritization
- 4 Future Work
- 5 Conclusions

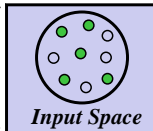


Approaches to Software Testing

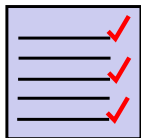
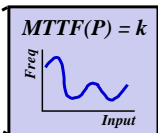


Structural Testing

Random Testing



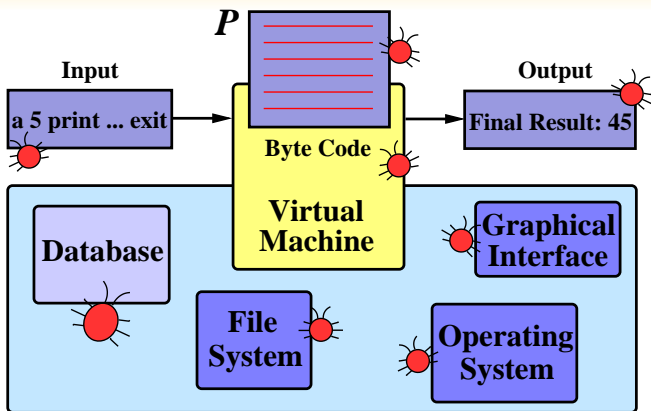
Software Reliability



Specification Testing

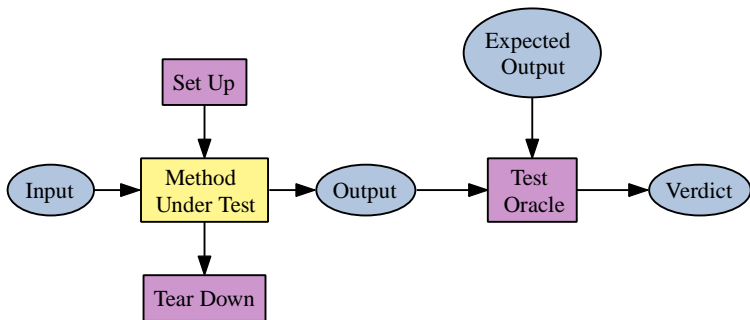
Testing **isolates defects** and establishes a **confidence in the correctness** of a software application

Where are Defects Located?



Defects may exist in the individual **components** or the **interactions**

What is a Test Case?



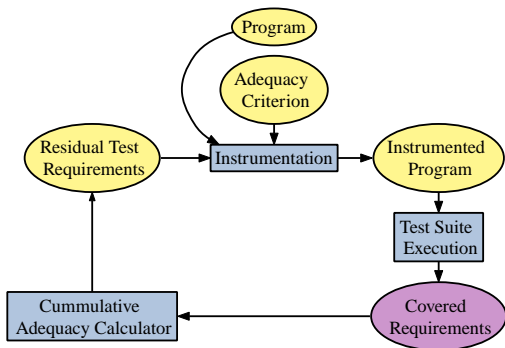
- Test suite executor runs each test case **independently**
- Each test invokes a method within the program and then compares the **actual** and **expected** output values

Using Tests to Find Faults

| Test Case | Faults | | | | |
|-----------|--------|-------|-------|-------|-------|
| | f_1 | f_2 | f_3 | f_4 | f_5 |
| T_1 | | | × | × | |
| T_2 | × | × | | | |
| T_3 | × | × | × | | |
| T_4 | | | × | × | × |
| T_5 | | × | × | | |

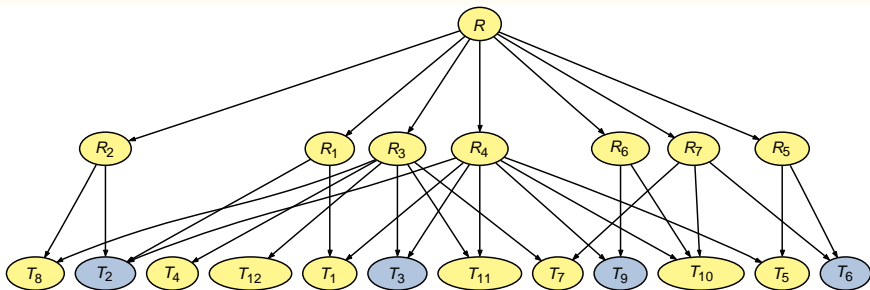
- **The Importance of Test Ordering:** $\langle T_3, T_4, T_1, T_2, T_5 \rangle$ detects faults more rapidly than $\langle T_1, T_2, T_3, T_4, T_5 \rangle$
- Since we do not have **a priori** knowledge of the faults that exist within a program, we must use a proxy like **coverage**

Test Coverage Monitoring



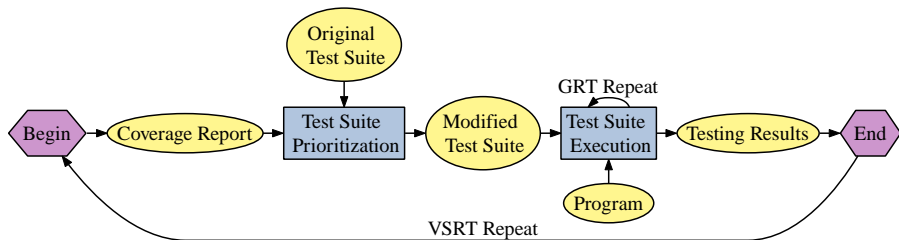
- Structural **adequacy criteria** focus on the coverage of nodes, edges, paths, and definition-use associations
- Instrumentation **probes** track the coverage of test requirements

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
- Test suite **reduction** discards the test cases that **redundantly** cover the test requirements

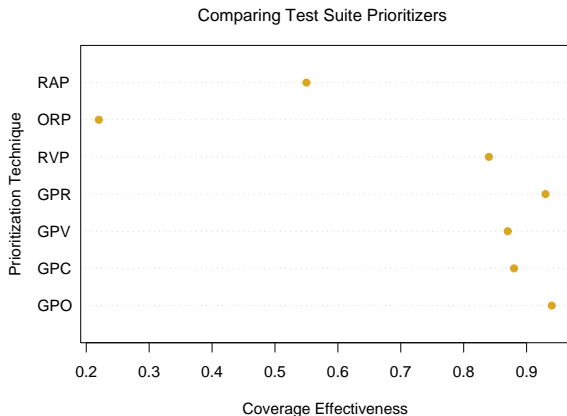
Regression Test Suite Prioritization



Overview

- Prioritization **re-orders** the tests so that they cover the requirements more effectively
- **GRT** uses the **same** prioritization across multiple runs of the test suite whereas **VSRT** creates a **new** prioritization for each test run

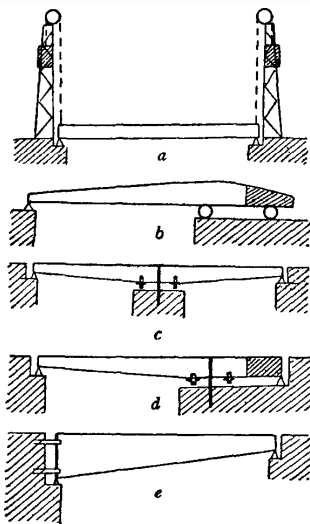
Comparing Prioritization Techniques



Which prioritization technique is the **best**?

Presentation Outline

- 1 Testing Challenges
- 2 Reduction and Prioritization
- 3 Time-Aware Prioritization**
- 4 Future Work
- 5 Conclusions



Time-Aware Orderings (Fault Data)

| | f_1 | f_2 | f_3 | f_4 | f_5 | f_6 | f_7 | f_8 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| T_1 | X | X | | X | X | X | X | X |
| T_2 | X | | | | | | | |
| T_3 | X | | | | X | | | |
| T_4 | | X | X | | | | X | |
| T_5 | | | | X | | X | | X |
| T_6 | | X | | X | | X | | |

- It is very common to confront a testing **time budget**
- Question:** If fault information is **known** and there is a testing time **limit**, then what is the **best** ordering?

Time-Aware Orderings (Faults and Costs)

| | # Faults | Time Cost | Avg. Faults per Min. |
|-------|----------|-----------|----------------------|
| T_1 | 7 | 9 | 0.778 |
| T_2 | 1 | 1 | 1.000 |
| T_3 | 2 | 3 | 0.667 |
| T_4 | 3 | 4 | 0.750 |
| T_5 | 3 | 4 | 0.750 |
| T_6 | 3 | 4 | 0.750 |

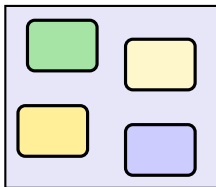
- When test case cost **varies**, then some tests are able to detect fault more **rapidly** than the others
- **Question:** What is the best ordering for this test suite?

Time-Aware Orderings (Comparison)

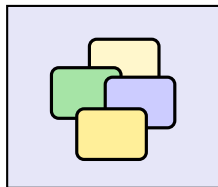
| Time Limit: 12 minutes | | | | |
|------------------------|-------|-------|-------|-------------|
| | Fault | Time | APFD | Intelligent |
| | T_1 | T_2 | T_2 | T_5 |
| | | T_3 | T_1 | T_4 |
| | | T_4 | | T_3 |
| | | T_5 | | |
| Tot. Faults | 7 | 8 | 7 | 8 |
| Tot. Time | 9 | 12 | 10 | 11 |

- The **existence** of a time limit **prevents** the use of traditional minimal set cover solvers that only look at **overlap**
- When fault information is **not** available, we can use **coverage**

Comparing Time-Aware Prioritizers



Traditional 0/1 Knapsack Solvers

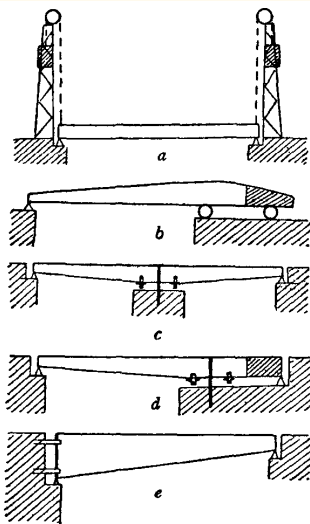


Genetic Algorithm Techniques

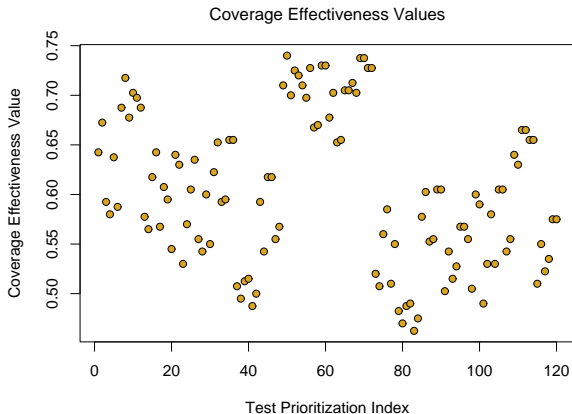
Empirical Results: prioritizers that consider coverage **overlap** take **longer** to re-order, but they arrive at **good** orderings

Presentation Outline

- 1 Testing Challenges
- 2 Reduction and Prioritization
- 3 Time-Aware Prioritization
- 4 Future Work**
- 5 Conclusions

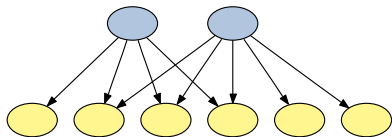


Search-Based Test Suite Prioritization

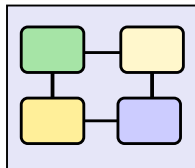


Use **heuristic search** (HC, SANN, GA) to prioritize the test suite

Detailed Empirical Evaluations



Synthetic Test Suites

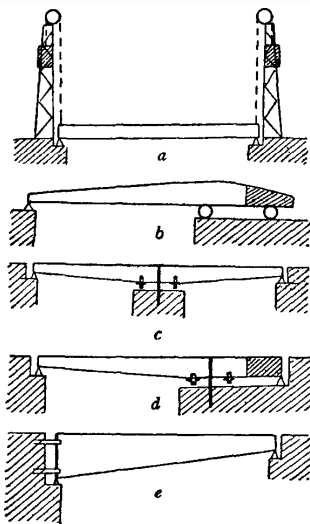


Real World Programs

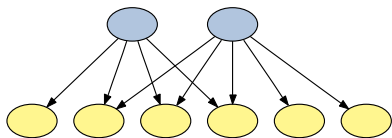
Conduct experiments to **systematically** study the trade-offs associated with testing techniques by using both **synthetic** coverage and **real world** applications

Presentation Outline

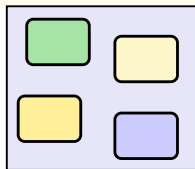
- 1 Testing Challenges
- 2 Reduction and Prioritization
- 3 Time-Aware Prioritization
- 4 Future Work
- 5 Conclusions



Concluding Remarks



Minimal Set Cover



0/1 Knapsack

- Establishes a **connection** between **practical** regression testing challenges and **NP-complete** problems
- Many approaches to testing are now **ready** for integration into frameworks such as **JUnit**

Personal Reflections

So now, come back to your God! Act on the principles of love and justice, and always live in confident dependence on your God.

Hosea 12:6 (New Living Translation)

Please keep in touch!

<http://www.cs.alleggheny.edu/~gkapfham/>