An Empirical Study of Incorporating Cost into Test Suite Reduction and Prioritization

[‡]Adam M. Smith and [†] Gregory M. Kapfhammer

[‡]Department of Computer Science
University of Pittsburgh

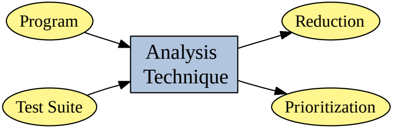
[†]Department of Computer Science
Allegheny College

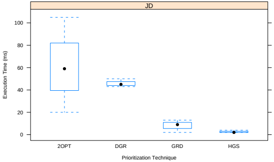
ACM Symposium on Applied Computing

March 8 - 12, 2009

Featuring an image from www.campusbicycle.com

Important Contributions



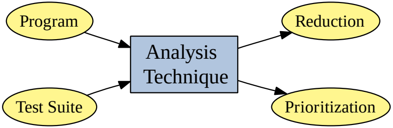


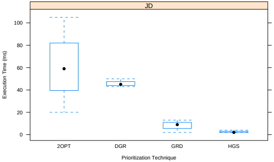
Regression Testing Techniques

Detailed Empirical Results

Implement and empirically **evaluate** the efficiency and effectiveness of **cost-aware** greedy methods for regression test suite **reduction** and **prioritization**

Important Contributions



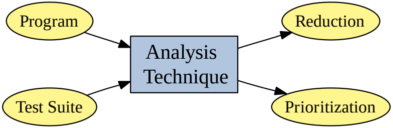


Regression Testing Techniques

Detailed Empirical Results

Implement and empirically **evaluate** the efficiency and effectiveness of **cost-aware** greedy methods for regression test suite **reduction** and **prioritization**

Important Contributions



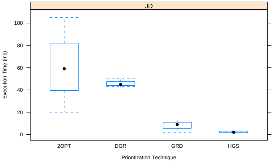
Program

Test Suite

Analysis Technique

Reduction

Prioritization

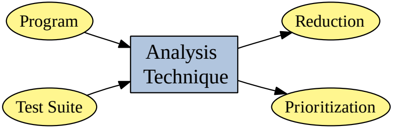


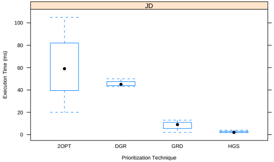
Regression Testing Techniques

Detailed Empirical Results

Implement and empirically **evaluate** the efficiency and effectiveness of **cost-aware** greedy methods for regression test suite **reduction** and **prioritization**

Important Contributions



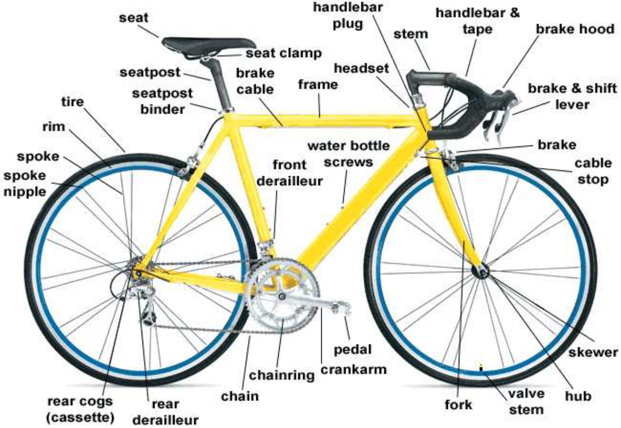


Regression Testing Techniques

Detailed Empirical Results

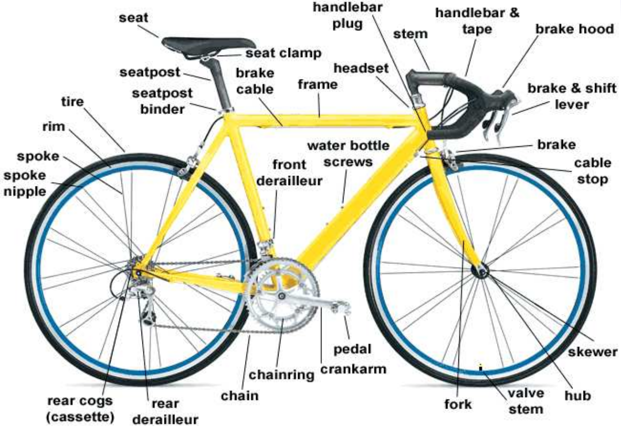
Implement and empirically **evaluate** the efficiency and effectiveness of **cost-aware** greedy methods for regression test suite **reduction** and **prioritization**

Regression Testing and Bicycles



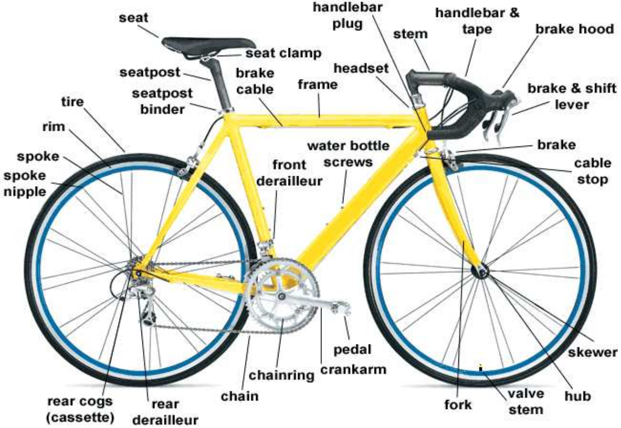
Efficiency: Low wind resistance and time to destination

Regression Testing and Bicycles



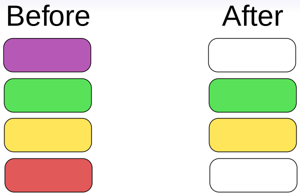
Effectiveness: Transports all required materials and no break downs

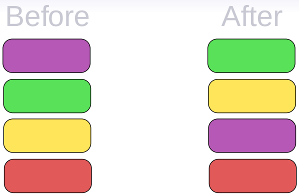
Regression Testing and Bicycles



Cost: Frame material and components cause price to vary considerably

Regression Testing Techniques



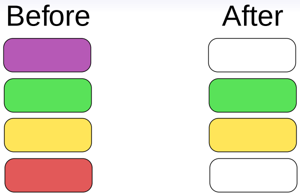


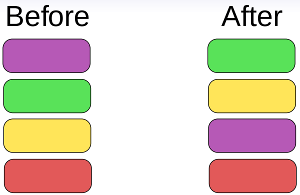
Reduction Prunes the Test Suite

Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the n tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection.**

Regression Testing Techniques



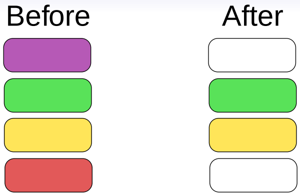


Reduction Prunes the Test Suite

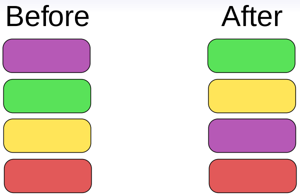
Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the n tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.

Regression Testing Techniques



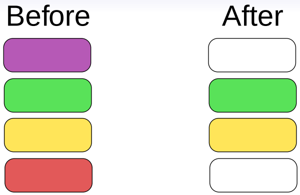
Reduction Prunes the Test Suite



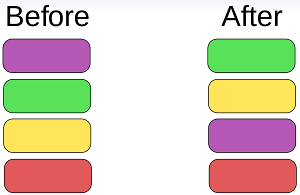
Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the n tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.

Regression Testing Techniques



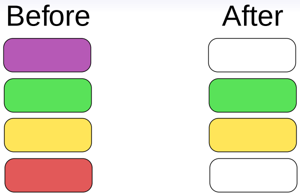
Reduction Prunes the Test Suite

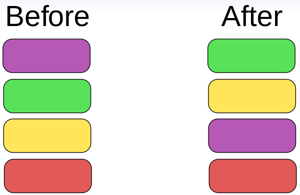


Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Reduction** discards some of the n tests in an attempt to **decrease** testing time while still **preserving** objectives like **coverage** or **fault detection**.

Regression Testing Techniques



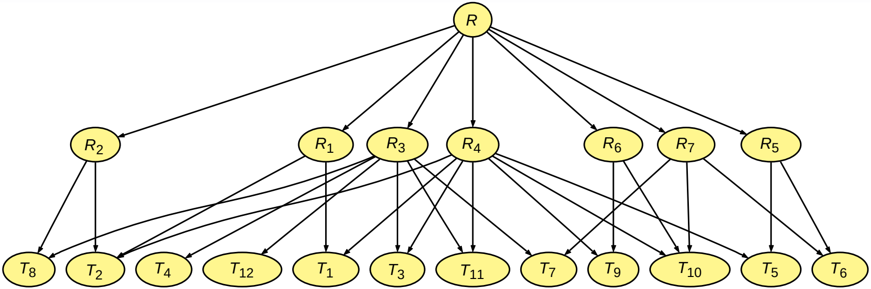


Reduction Prunes the Test Suite

Prioritization Reorders the Tests

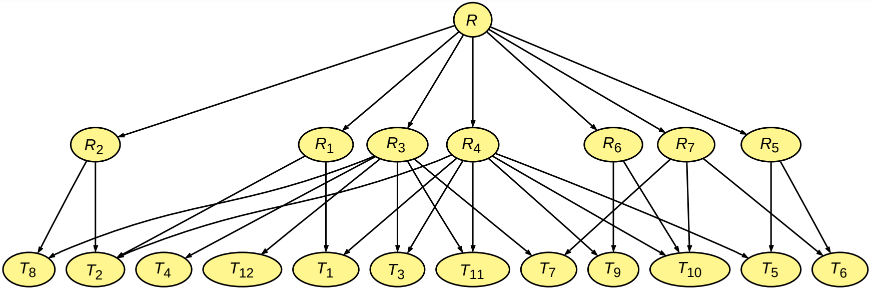
It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Prioritization** searches through the $n! = n \times n - 1 \times \ldots \times 1$ orderings for those that **maximize** an objective function like **coverage** or **fault detection.**

Finding the Overlap in Coverage



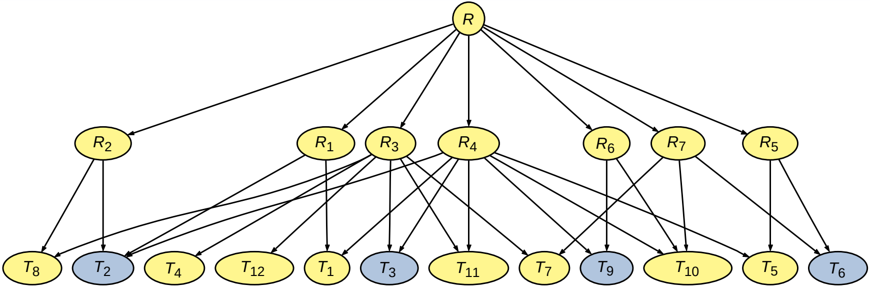
- $R_j \rightarrow T_i$ means that requirement R_j is **covered by** test T_i
- Test suite **reduction** discards the test cases that **redundantly** cover the test requirements
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements

Finding the Overlap in Coverage



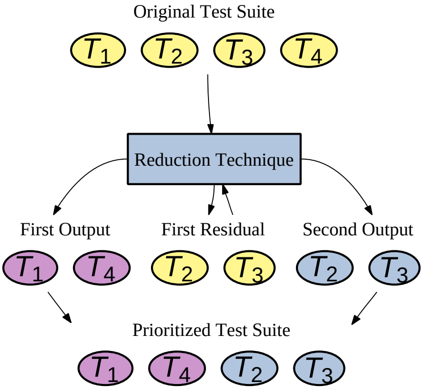
- $R_j \rightarrow T_i$ means that requirement R_j is **covered by** test T_i
- Test suite **reduction** discards the test cases that **redundantly** cover the test requirements
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements

Finding the Overlap in Coverage



- $R_j \rightarrow T_i$ means that requirement R_j is **covered by** test T_i
- Test suite **reduction** discards the test cases that **redundantly** cover the test requirements
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements

Greedy Approaches to Regression Testing

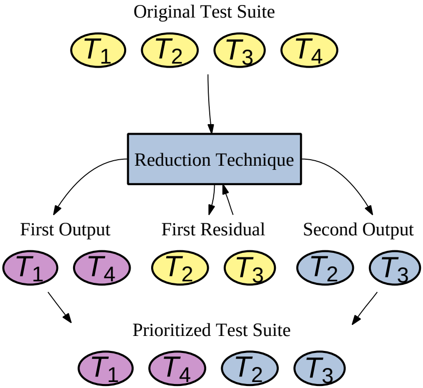


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

Hypothesis: Using the execution **time** of a test case can **improve** the reduced and prioritized test suites

Compare (i) **greedy choices** (cost, coverage, and ratio) and (ii) **algorithms**

Greedy Approaches to Regression Testing

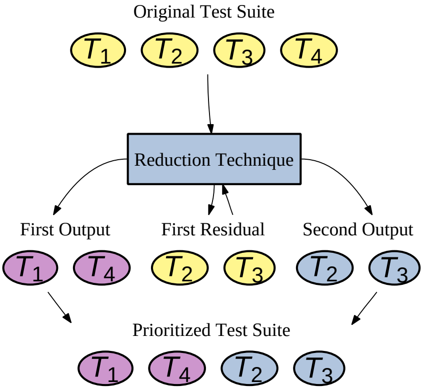


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

Hypothesis: Using the execution **time** of a test case can **improve** the reduced and prioritized test suites

Compare (i) **greedy choices** (cost, coverage, and ratio) and (ii) **algorithms**

Greedy Approaches to Regression Testing

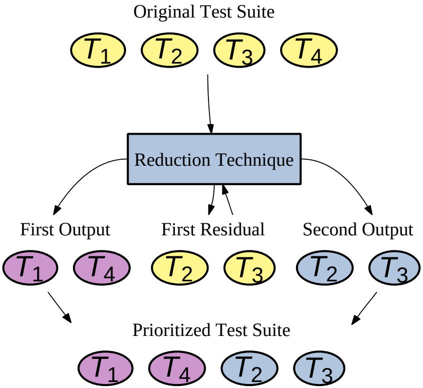


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

Hypothesis: Using the execution **time** of a test case can **improve** the reduced and prioritized test suites

Compare (i) **greedy choices** (cost, coverage, and ratio) and (ii) **algorithms**

Greedy Approaches to Regression Testing

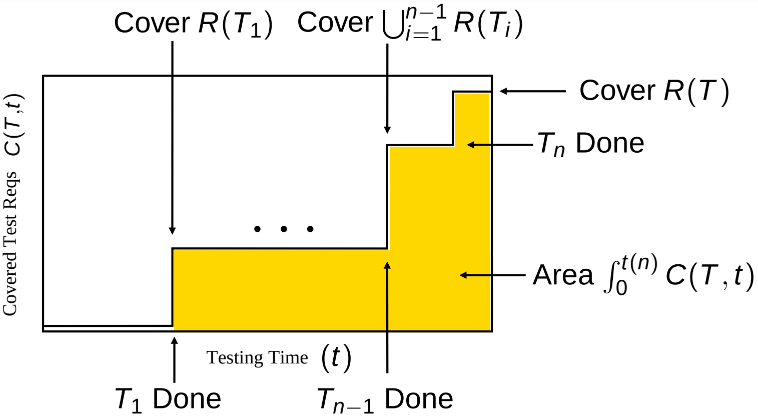


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

Hypothesis: Using the execution **time** of a test case can **improve** the reduced and prioritized test suites

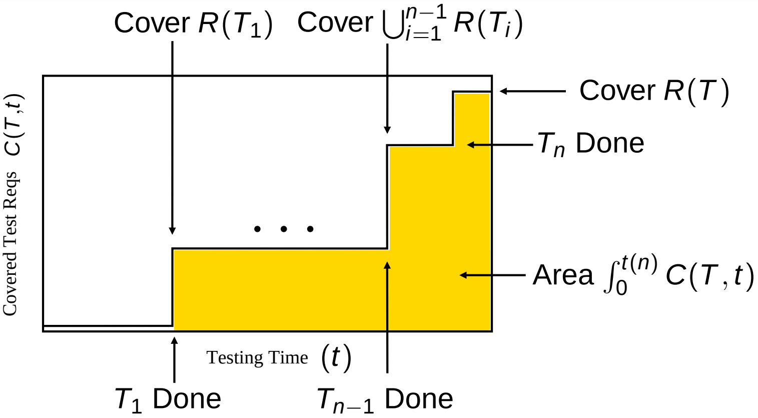
Compare (i) **greedy choices** (cost, coverage, and ratio) and (ii) **algorithms**

Evaluating Test Suite Prioritizers


Cover $R(T_1)$
Cover $\bigcup_{i=1}^{n-1} R(T_i)$
Cover $R(T)$
T_n Done
. . .
Area $\int_0^{t(n)} C(T,t)$
Covered Test Reqs $C(T,t)$
Testing Time (t)
T_1 Done
T_{n-1} Done

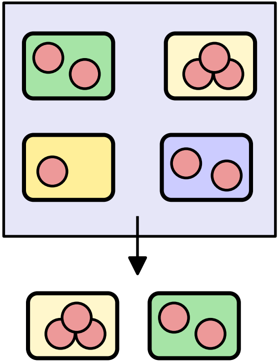
- Prioritize to **increase** the CE of a test suite $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0,1]$

Evaluating Test Suite Prioritizers


Cover $R(T_1)$
Cover $\bigcup_{i=1}^{n-1} R(T_i)$
Cover $R(T)$
T_n Done
Area $\int_0^{t(n)} C(T,t)$
Covered Test Reqs $C(T,t)$
Testing Time (t)
T_1 Done
T_{n-1} Done
. . .

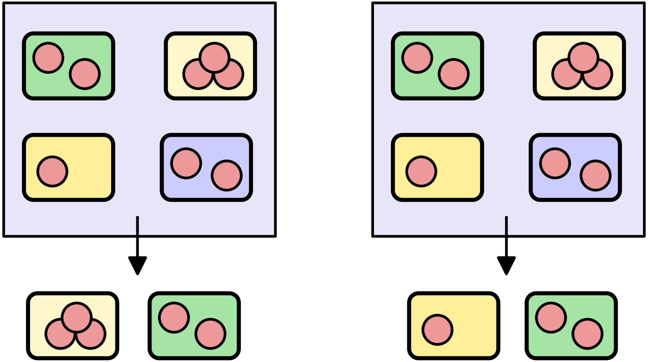
- Prioritize to **increase** the CE of a test suite $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0, 1]$

Evaluating Test Suite Reducers



Reduction Factor for Size (RFFS): How small is the reduced test suite?

Evaluating Test Suite Reducers



Reduction Factor for Time (RFFT): How fast is the reduced test suite?

Greedy Choices Impact Effectiveness

	R_1	R_2	R_3	R_4	R_5	Execution Time
T_1	✓	✓	✓	✓		4
T_2			✓	✓		1
T_3		✓				1
T_4	✓				✓	1

Greedy-by	T_r	$time(T_r)$	T_p	CE
coverage	$\langle T_1, T_4 \rangle$	5	$\langle T_1, T_4, T_2, T_3 \rangle$	0.400
time	$\langle T_2, T_3, T_4 \rangle$	3	$\langle T_2, T_3, T_4, T_1 \rangle$	0.714
ratio	$\langle T_2, T_4, T_3 \rangle$	3	$\langle T_2, T_4, T_3, T_1 \rangle$	0.743

Greedy Choices Impact Effectiveness

	R_1	R_2	R_3	R_4	R_5	Execution Time
T_1	✓	✓	✓	✓		4
T_2			✓	✓		1
T_3		✓				1
T_4	✓				✓	1

Greedy-by	T_r	$time(T_r)$	T_p	CE
coverage	$\langle T_1, T_4 \rangle$	5	$\langle T_1, T_4, T_2, T_3 \rangle$	0.400
time	$\langle T_2, T_3, T_4 \rangle$	3	$\langle T_2, T_3, T_4, T_1 \rangle$	0.714
ratio	$\langle T_2, T_4, T_3 \rangle$	3	$\langle T_2, T_4, T_3, T_1 \rangle$	0.743

Greedy Choices Impact Effectiveness

	R_1	R_2	R_3	R_4	R_5	Execution Time
T_1	✓	✓	✓	✓		4
T_2			✓	✓		1
T_3		✓				1
T_4	✓				✓	1

Greedy-by	T_r	$time(T_r)$	T_p	CE
coverage	$\langle T_1, T_4 \rangle$	5	$\langle T_1, T_4, T_2, T_3 \rangle$	0.400
time	$\langle T_2, T_3, T_4 \rangle$	3	$\langle T_2, T_3, T_4, T_1 \rangle$	0.714
ratio	$\langle T_2, T_4, T_3 \rangle$	3	$\langle T_2, T_4, T_3, T_1 \rangle$	0.743

Greedy Choices Impact Effectiveness

	R_1	R_2	R_3	R_4	R_5	Execution Time
T_1	✓	✓	✓	✓		4
T_2			✓	✓		1
T_3		✓				1
T_4	✓				✓	1

Greedy-by	T_r	$time(T_r)$	T_p	CE
coverage	$\langle T_1, T_4 \rangle$	5	$\langle T_1, T_4, T_2, T_3 \rangle$	0.400
time	$\langle T_2, T_3, T_4 \rangle$	3	$\langle T_2, T_3, T_4, T_1 \rangle$	0.714
ratio	$\langle T_2, T_4, T_3 \rangle$	3	$\langle T_2, T_4, T_3, T_1 \rangle$	0.743

Case Study Applications

Name	$\|T\|$	$\|\mathcal{R}(T)\|$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Case Study Applications

Name	$\lvert T \rvert$	$\lvert \mathcal{R}(T) \rvert$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Case Study Applications

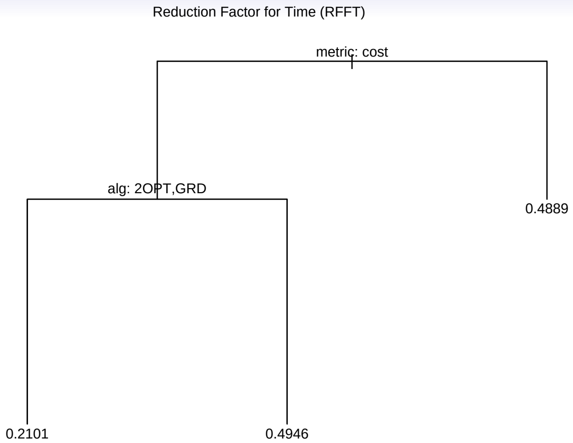
Name	$\lvert T \rvert$	$\lvert \mathcal{R}(T) \rvert$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

Case Study Applications

Name	$\lvert T \rvert$	$\lvert \mathcal{R}(T) \rvert$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

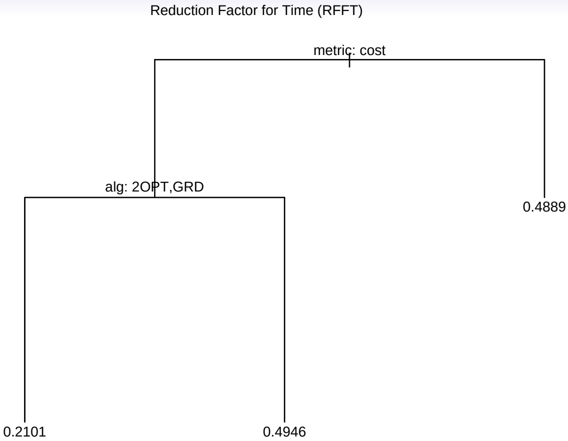
Questions: Do the **greedy** reducers and prioritizers efficiently identify test suites that **improve** effectiveness? What are the fundamental **trade-offs**?

Overview of RFFT Trends



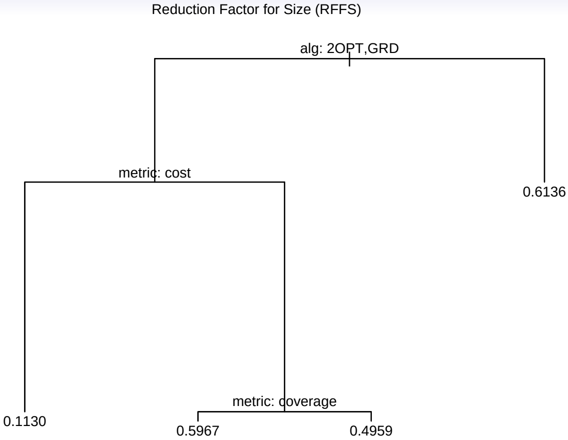
The myopic focus on **cost** leads to **low** RFFT values for 2OPT and GRD

Overview of RFFT Trends



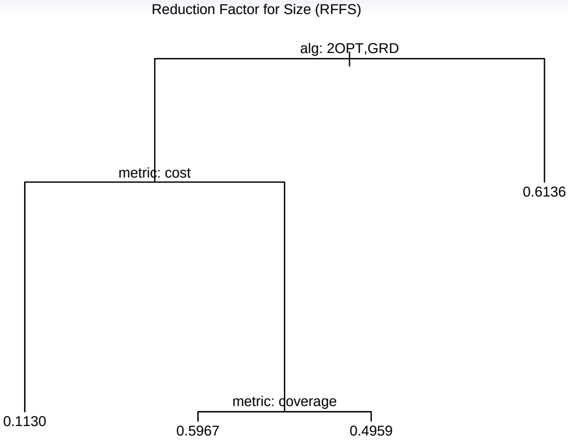
The myopic focus on **cost** leads to **low** RFFT values for 2OPT and GRD

Overview of RFFS Trends



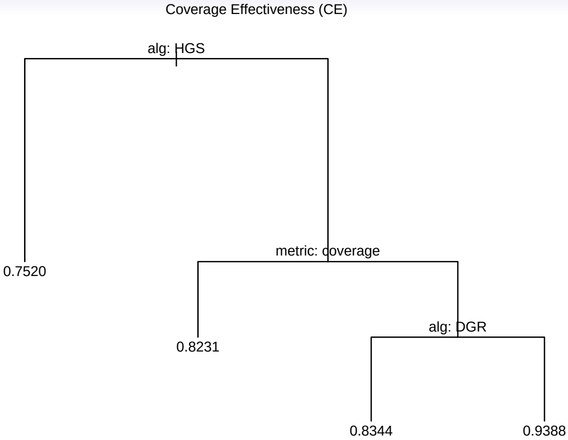
DGR and HGS are the **best** at creating test suites that **improve** RFFS

Overview of RFFS Trends



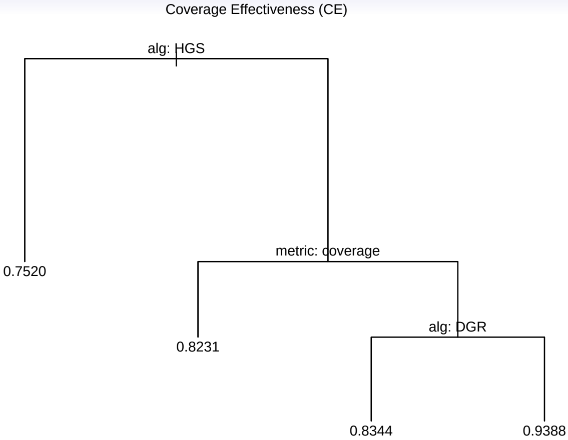
DGR and HGS are the **best** at creating test suites that **improve** RFFS

Overview of CE Trends



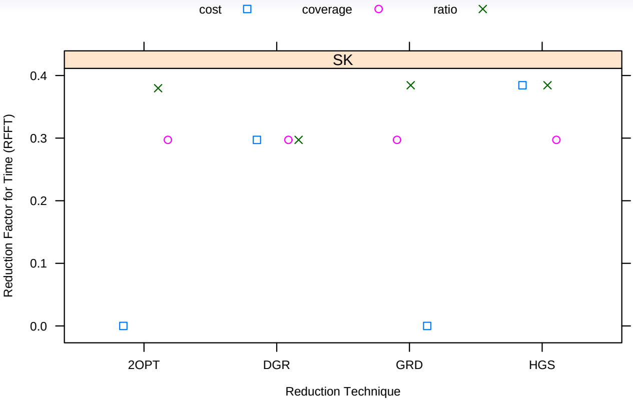
Using **ratio** and **cost** improves the CE of the prioritized test suite

Overview of CE Trends



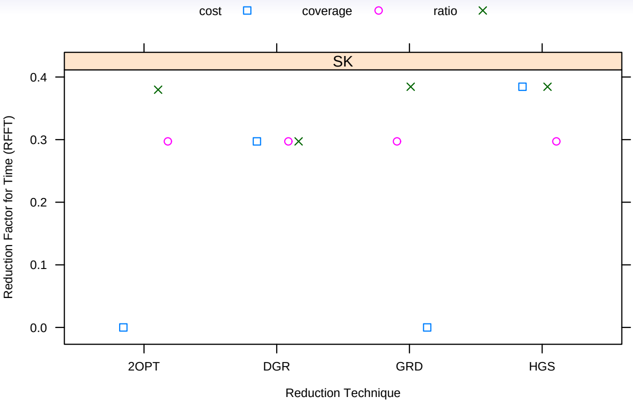
Using **ratio** and **cost** improves the CE of the prioritized test suite

Reduction Factor for Time - SK



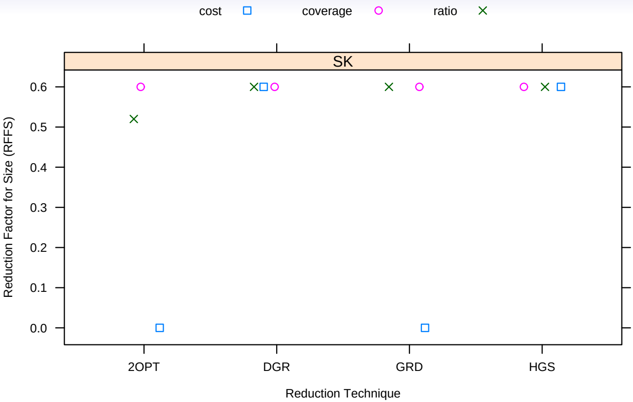
For 2OPT and GRD, **ratio** and **coverage** create the best test suites

Reduction Factor for Time - SK



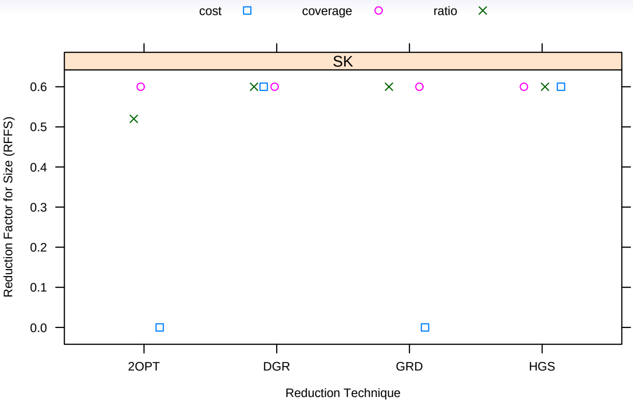
For 2OPT and GRD, **ratio** and **coverage** create the best test suites

Reduction Factor for Size - SK



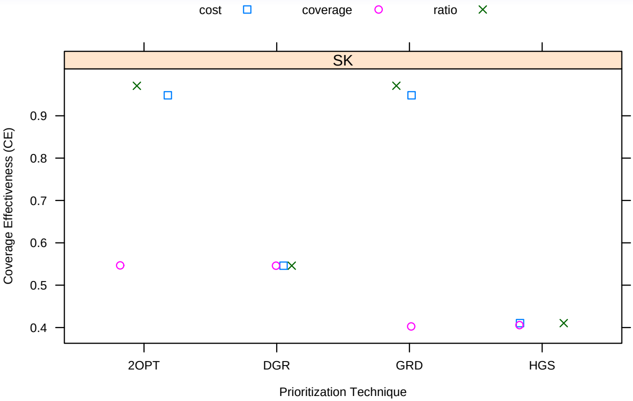
It is often **easy** to construct test suites with **high** RFFS values

Reduction Factor for Size - SK



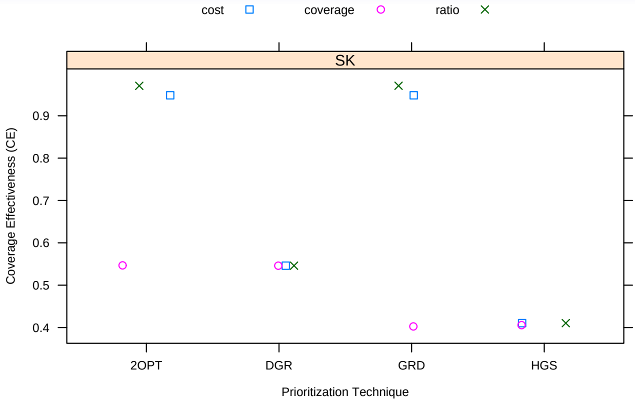
It is often **easy** to construct test suites with **high** RFFS values

Coverage Effectiveness Results - RP



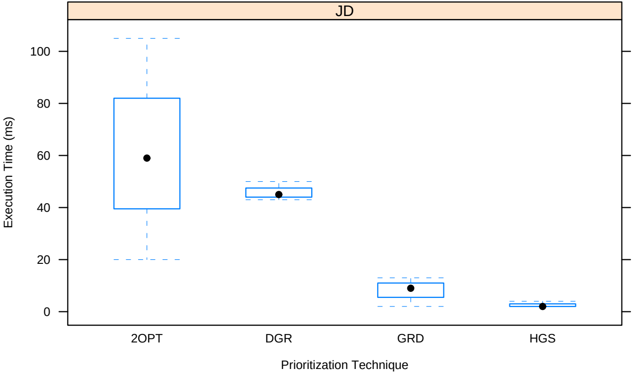
DGR and HGS exhibit **lackluster** performance when **reordering**

Coverage Effectiveness Results - RP



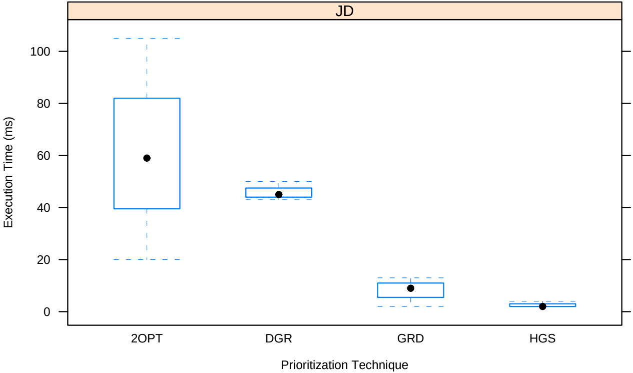
DGR and HGS exhibit **lackluster** performance when **reordering**

Efficiency Measurements



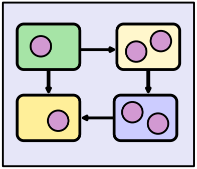
For the **chosen** case study applications, the techniques are **efficient**

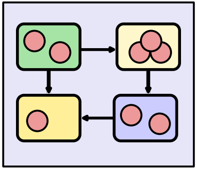
Efficiency Measurements



For the **chosen** case study applications, the techniques are **efficient**

Alternative Evaluation Metrics Like APFD



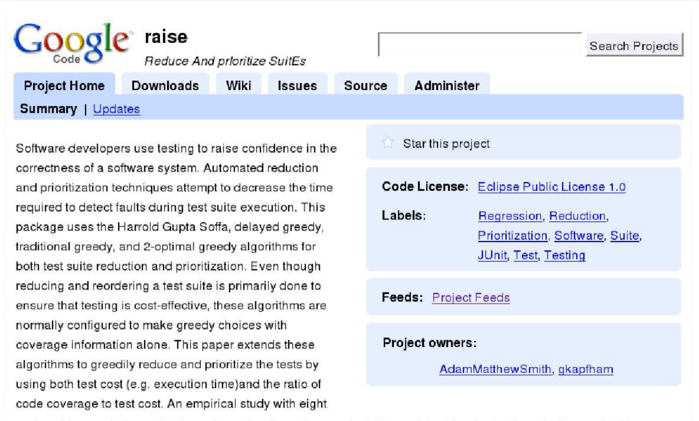


Mutation Faults

Real Faults

Use **mutation** and **real** faults to support the calculation of fault detection effectiveness (**FDE**) and average percentage of faults detected (**APFD**). Consider **search-based** testing methods.

RAISE - *R*educe *A*nd pr*I*ortize Suit*E*s



raise

Reduce And prIoritize SuitEs

Software developers use testing to raise confidence in the correctness of a software system. Automated reduction and prioritization techniques attempt to decrease the time required to detect faults during test suite execution. This package uses the Harrold Gupta Soffa, delayed greedy, traditional greedy, and 2-optimal greedy algorithms for both test suite reduction and prioritization. Even though reducing and reordering a test suite is primarily done to ensure that testing is cost-effective, these algorithms are normally configured to make greedy choices with coverage information alone. This paper extends these algorithms to greedily reduce and prioritize the tests by using both test cost (e.g. execution time)and the ratio of code coverage to test cost. An empirical study with eight

Code License: Eclipse Public License 1.0

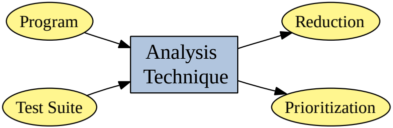
Labels: Regression, Reduction, Prioritization, Software, Suite, JUnit, Test, Testing

Feeds: Project Feeds

Project owners: AdamMatthewSmith, gkapfham

http://raise.googlecode.com/ provides tools, data sets, and resources

Concluding Remarks



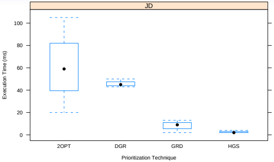
Program

Test Suite

Analysis Technique

Reduction

Prioritization



Regression Testing Techniques

Detailed Empirical Results

- **Implementation** and empirical **evaluation** of methods for test suite reduction and prioritization
- Freely available **data sets** and free/open source **tools**

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