Practical Suggestions for Improving and Empirically Studying Greedy Test Suite Reduction and Prioritization Methods

Gregory M. Kapfhammer†

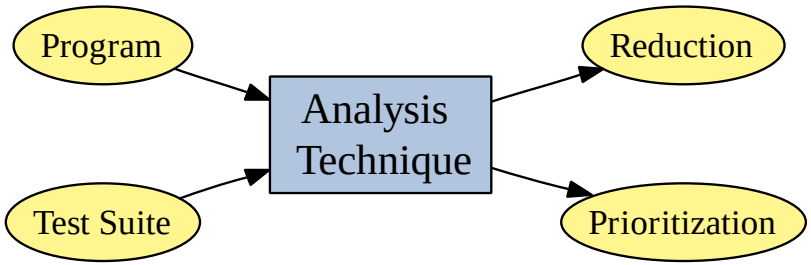
Department of Computer Science
Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

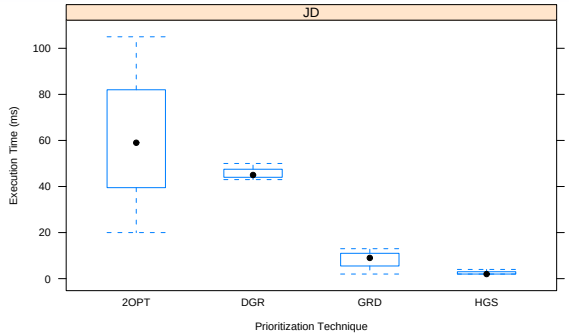
Department of Computer Science and Technology
Nanjing University, November 2009

† In Conjunction with Adam M. Smith, Joshua J. Geiger, G. Elisabeta Mara (University of Pittsburgh)
Manos Renieris (Google)

Featuring an image from www.CampusBicycle.com

Important Contributions



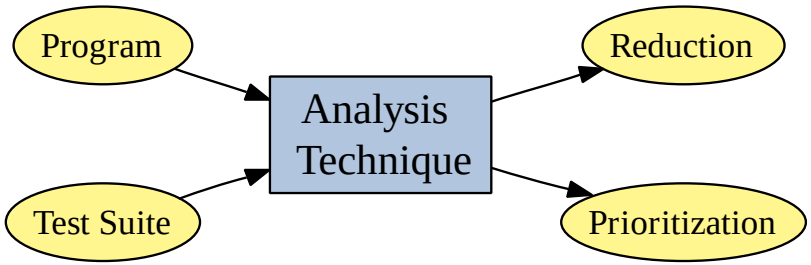


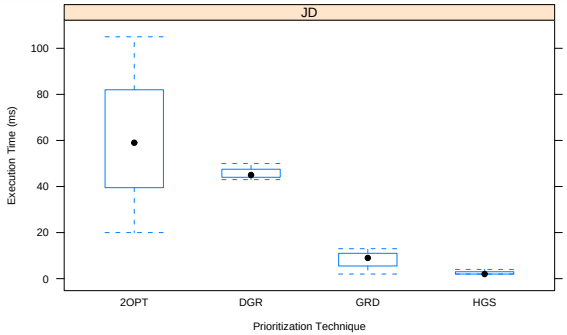
Regression Testing Techniques

Detailed Empirical Study

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

Important Contributions



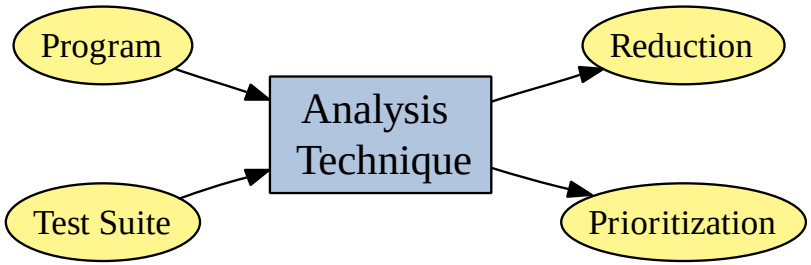


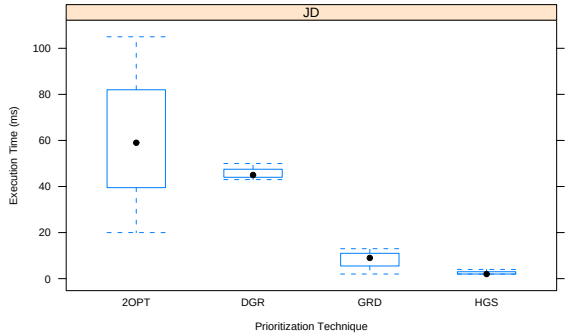
Regression Testing Techniques

Detailed Empirical Study

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

Important Contributions



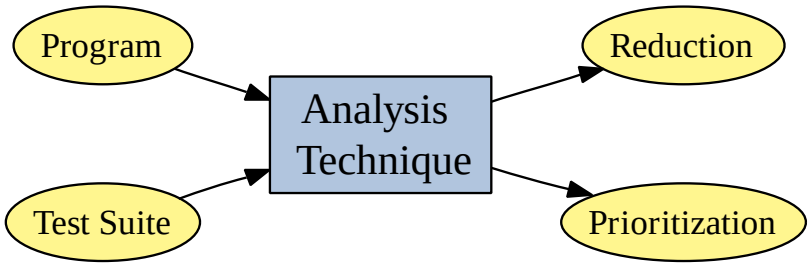


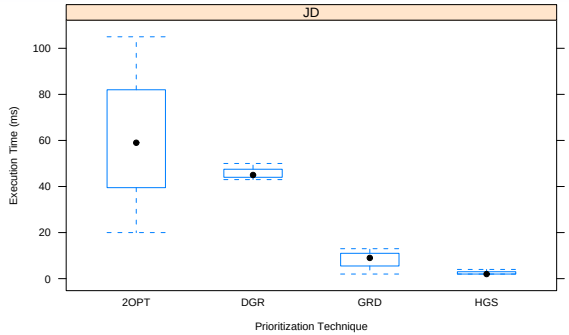
Regression Testing Techniques

Detailed Empirical Study

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

Important Contributions



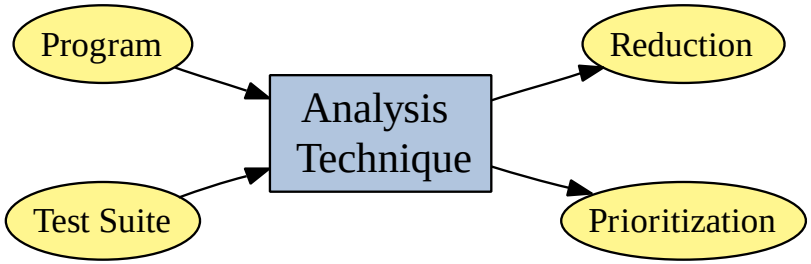


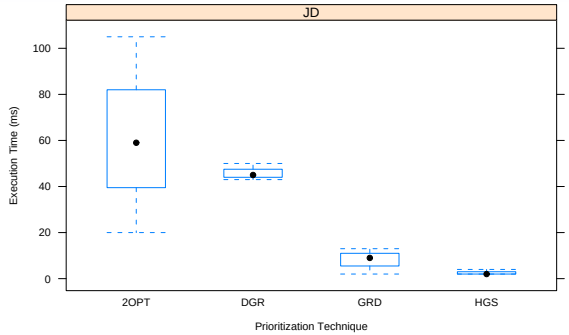
Regression Testing Techniques

Detailed Empirical Study

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

Important Contributions



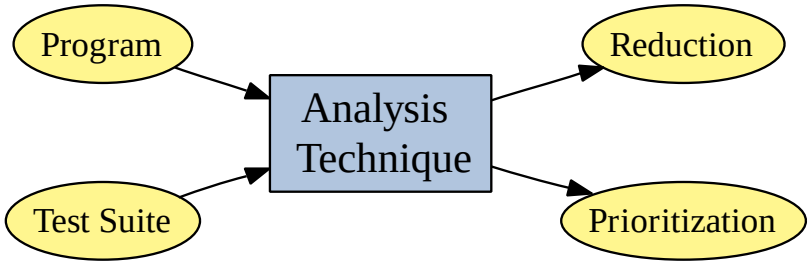


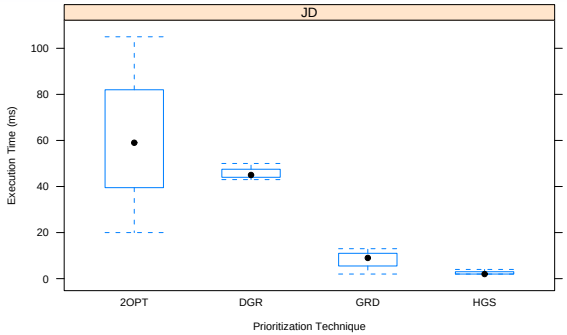
Regression Testing Techniques

Detailed Empirical Study

Experiments: Use automatically generated **synthetic** test suites and **real world** case study applications during the **empirical** study of greedy **regression testing** methods

Important Contributions



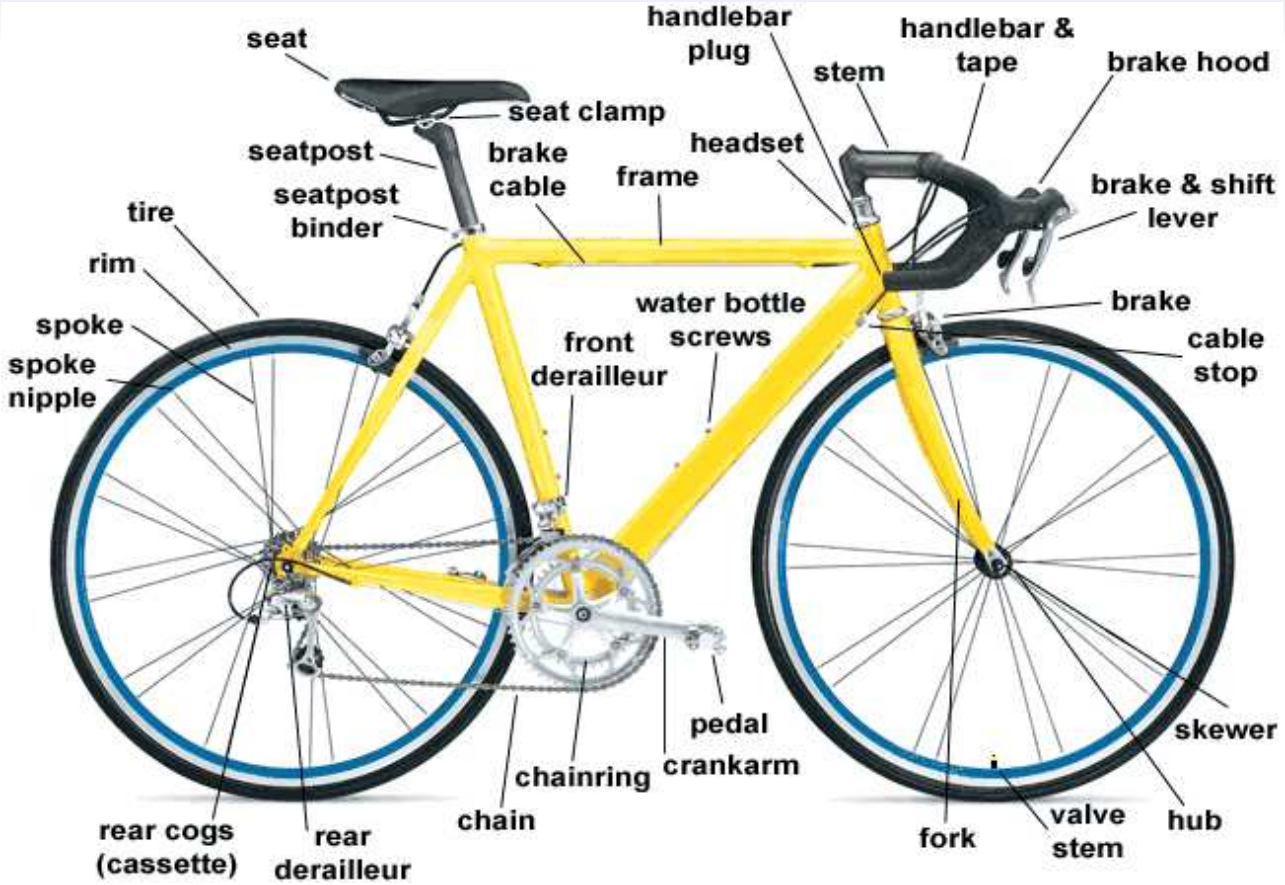


Regression Testing Techniques

Detailed Empirical Study

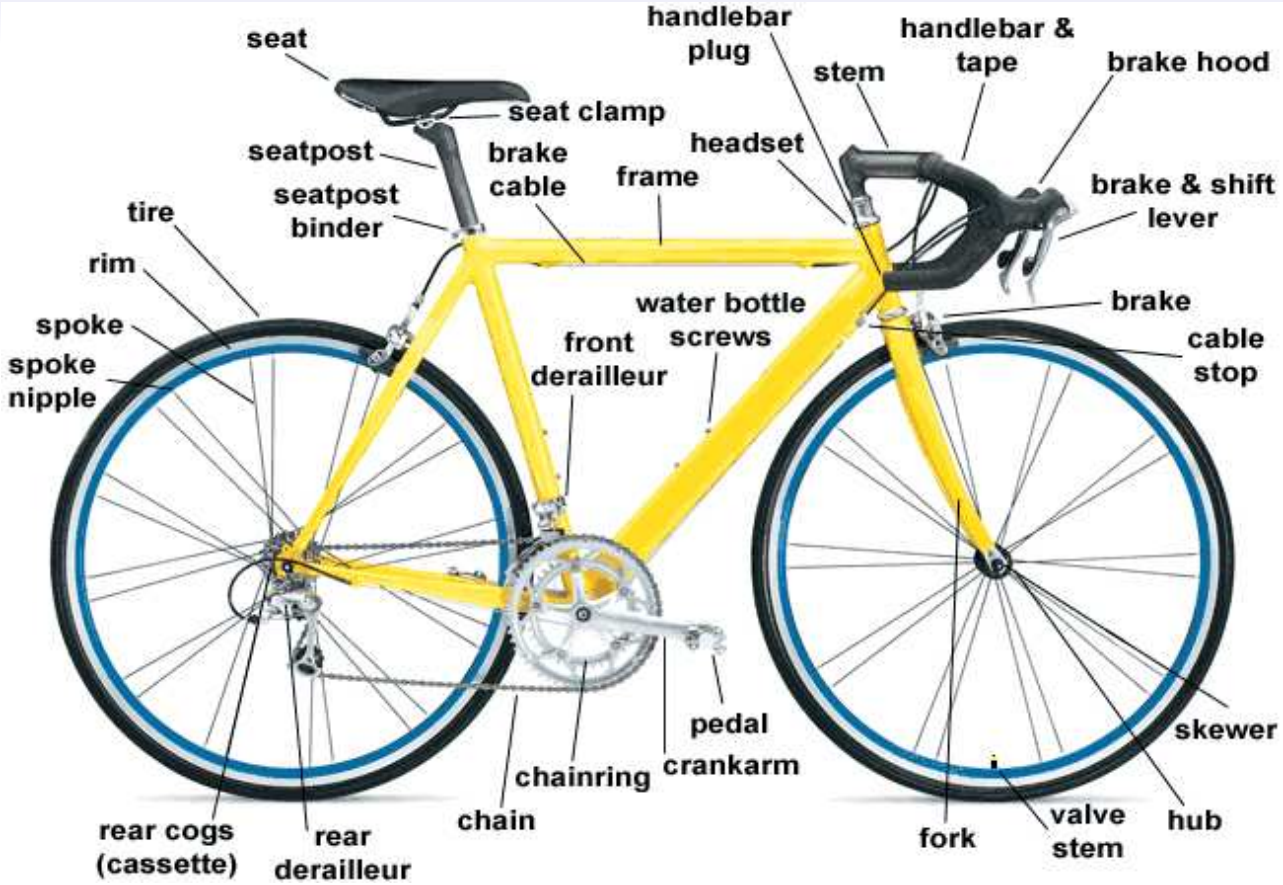
Analysis: Develop and use **tree** and **random forest** statistical models and interactive **visualization** techniques that help to **identify** efficiency and effectiveness **trade-offs** for testing

Regression Testing and Bicycles



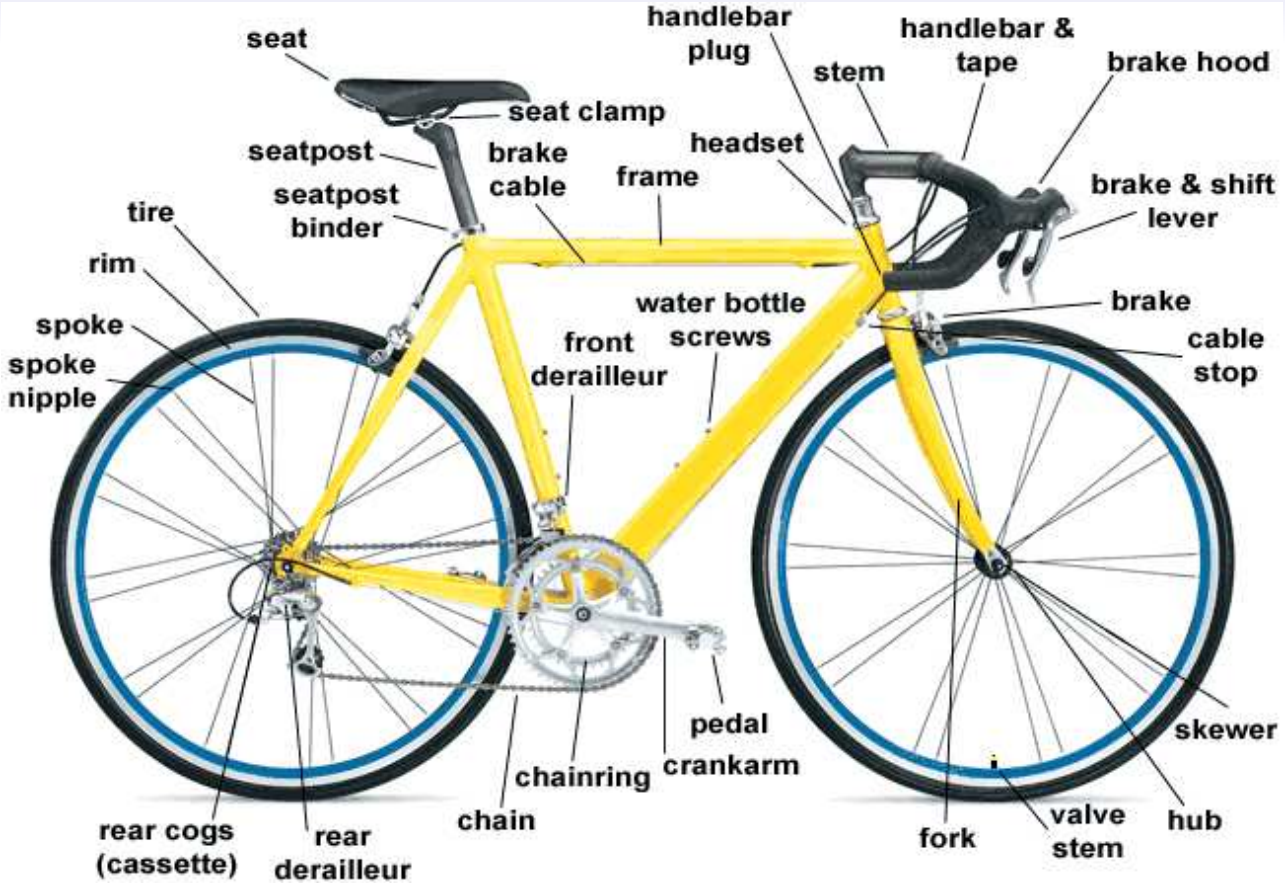
Effi ciency: Low wind resistance and time to destination

Regression Testing and Bicycles



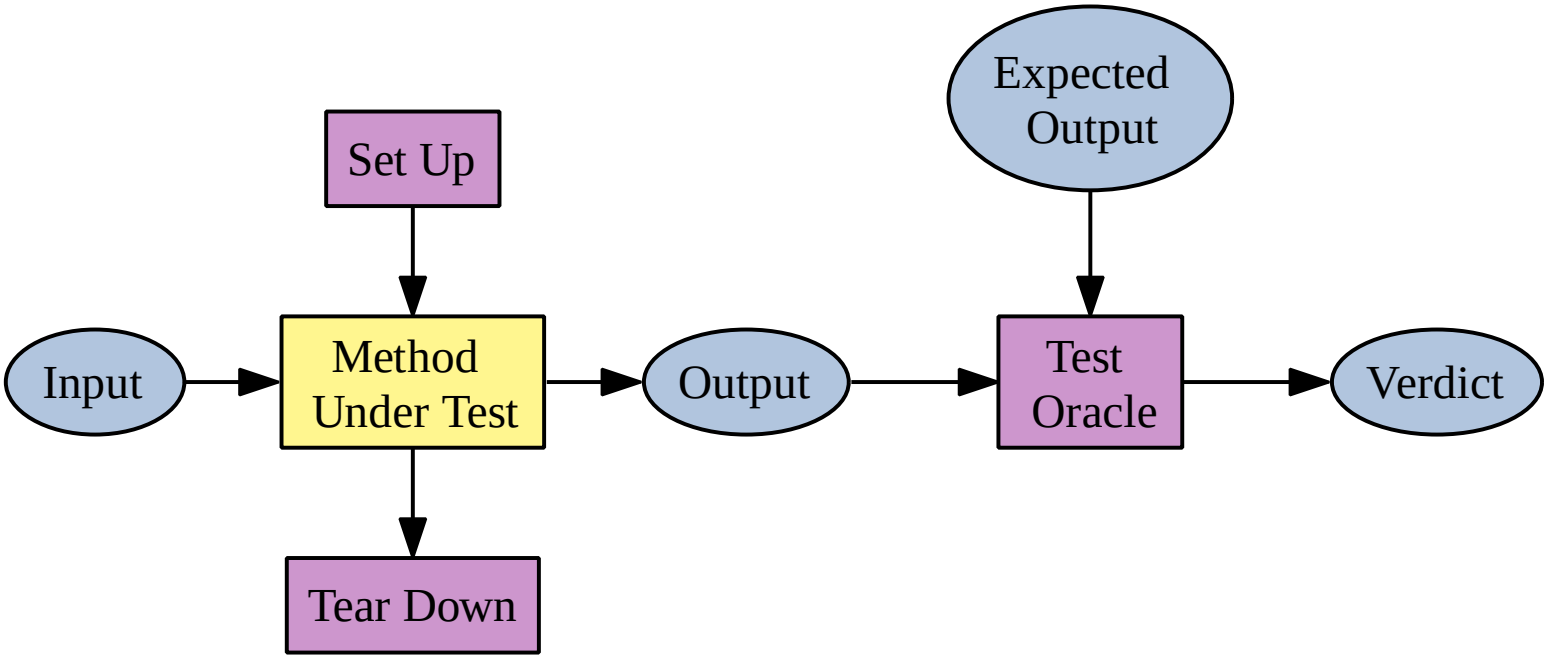
Effectiveness: Transports all required materials and no break downs

Regression Testing and Bicycles



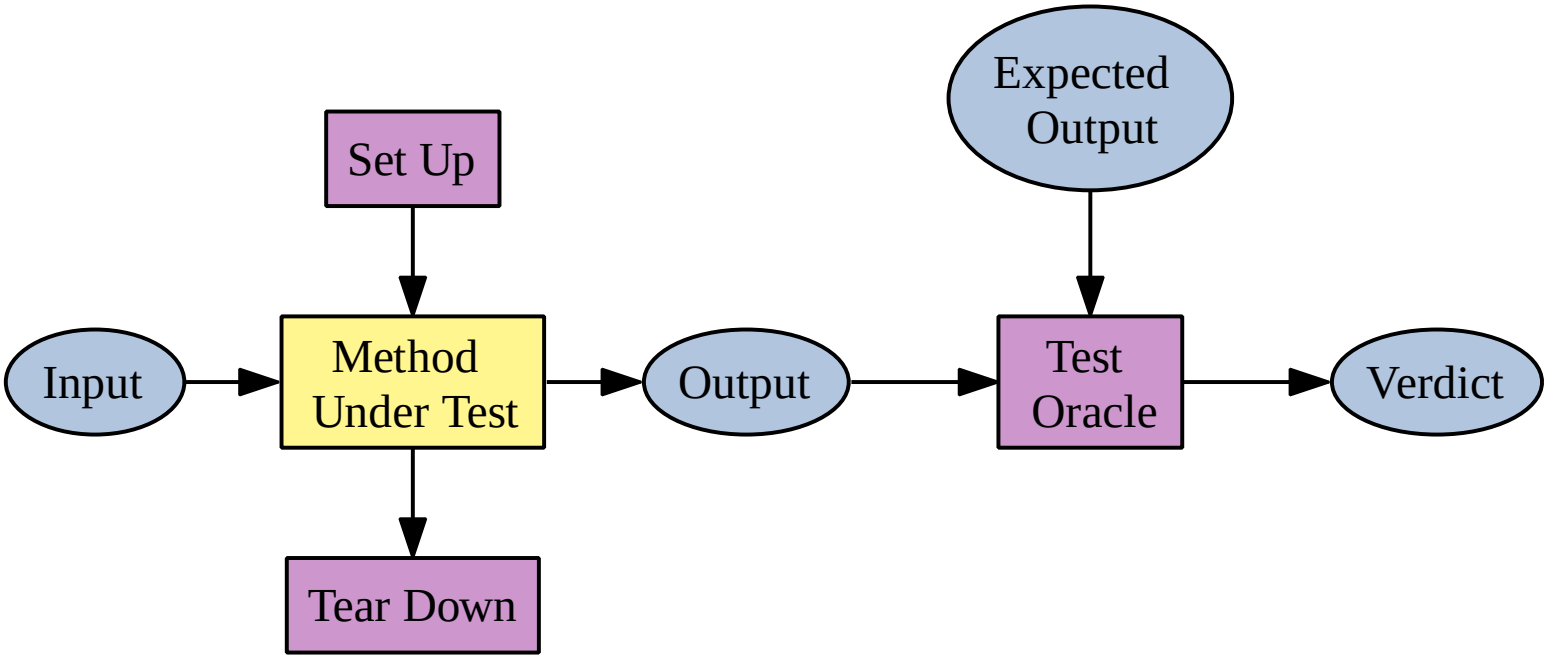
Cost: Frame material and components cause price to vary considerably

What is a Test Case?



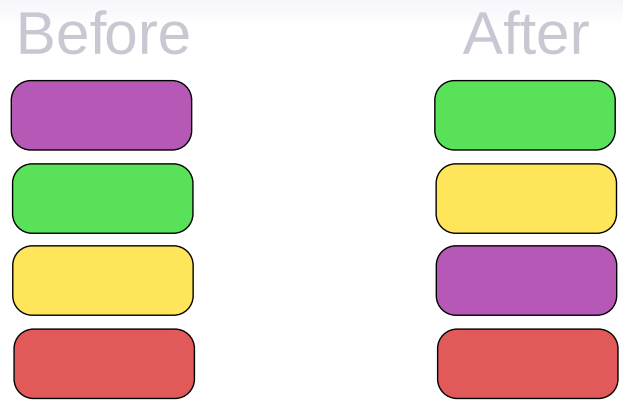
- Test suite **executor** runs each test case **independently**
- Each test invokes a **method** within the program and then compares the **actual** and **expected** output values

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

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**.

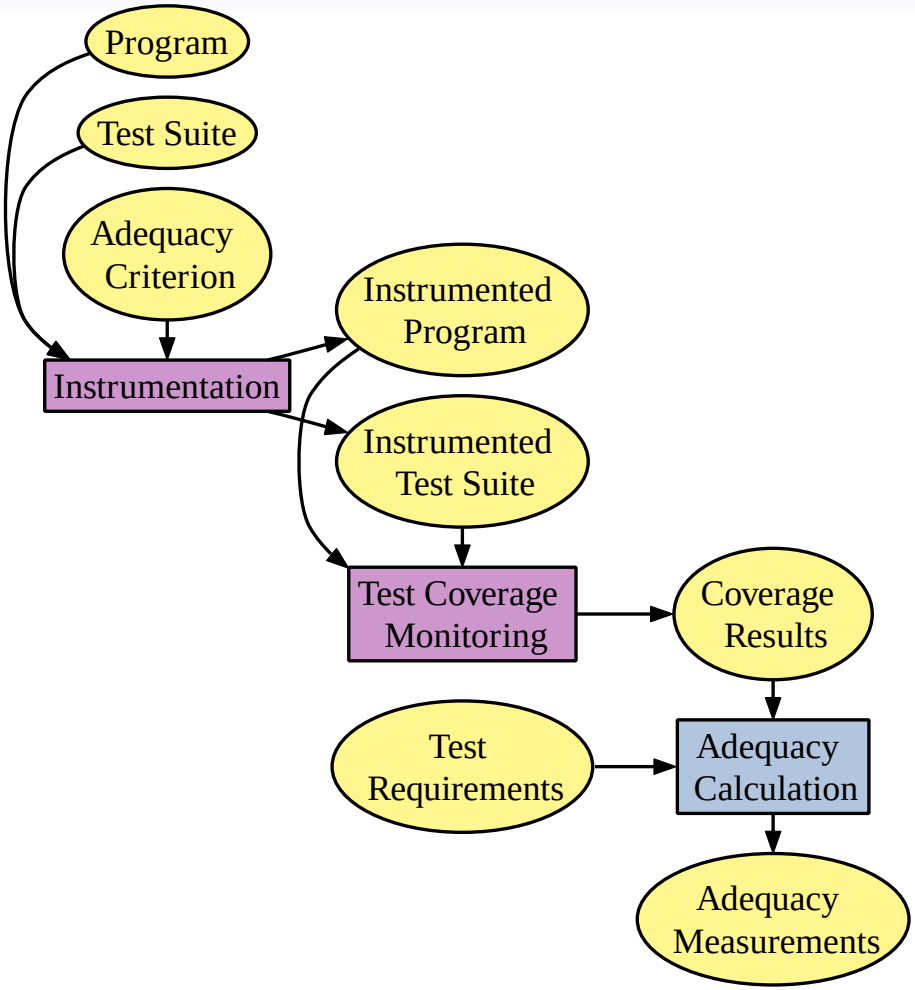
Regression Testing Techniques

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**.

Calculating the Coverage of a Test Suite



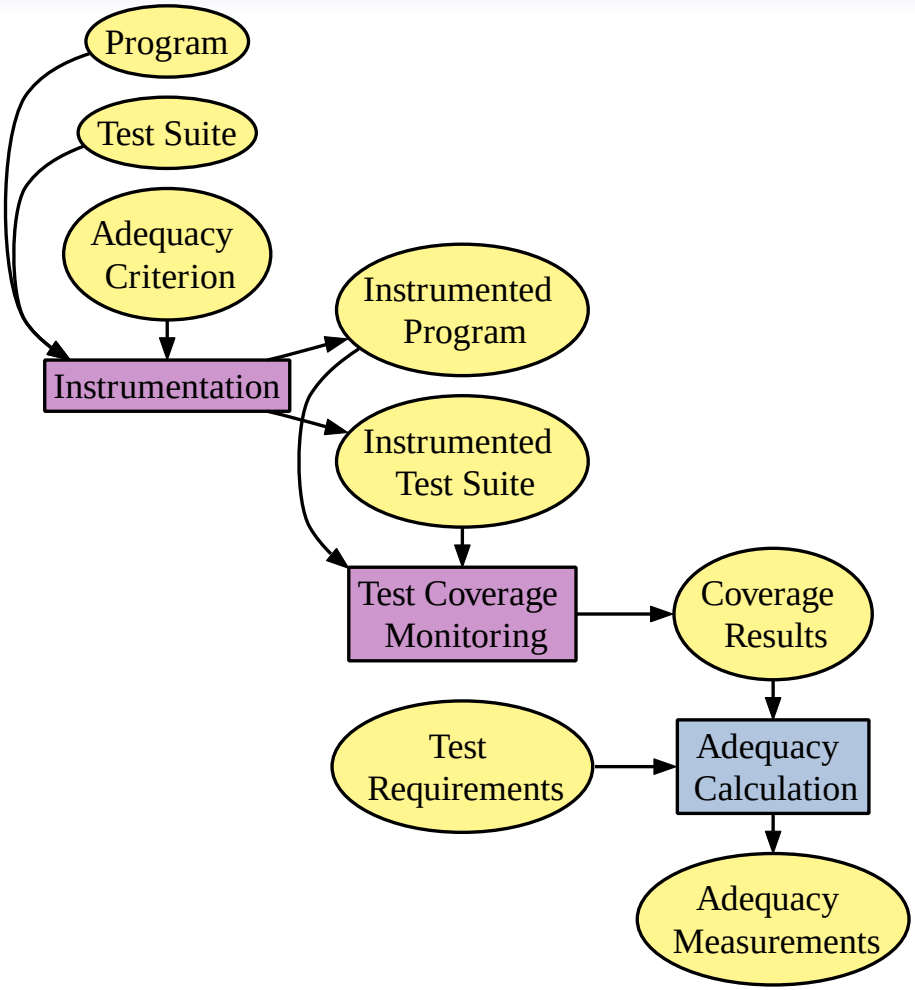
Calculating Coverage

Use instrumentation probes to **capture** and **analyze** a test suite's coverage of the program state and structure

Regression Testing

The coverage results and adequacy measurements can support both test suite **reduction** and **prioritization**

Calculating the Coverage of a Test Suite



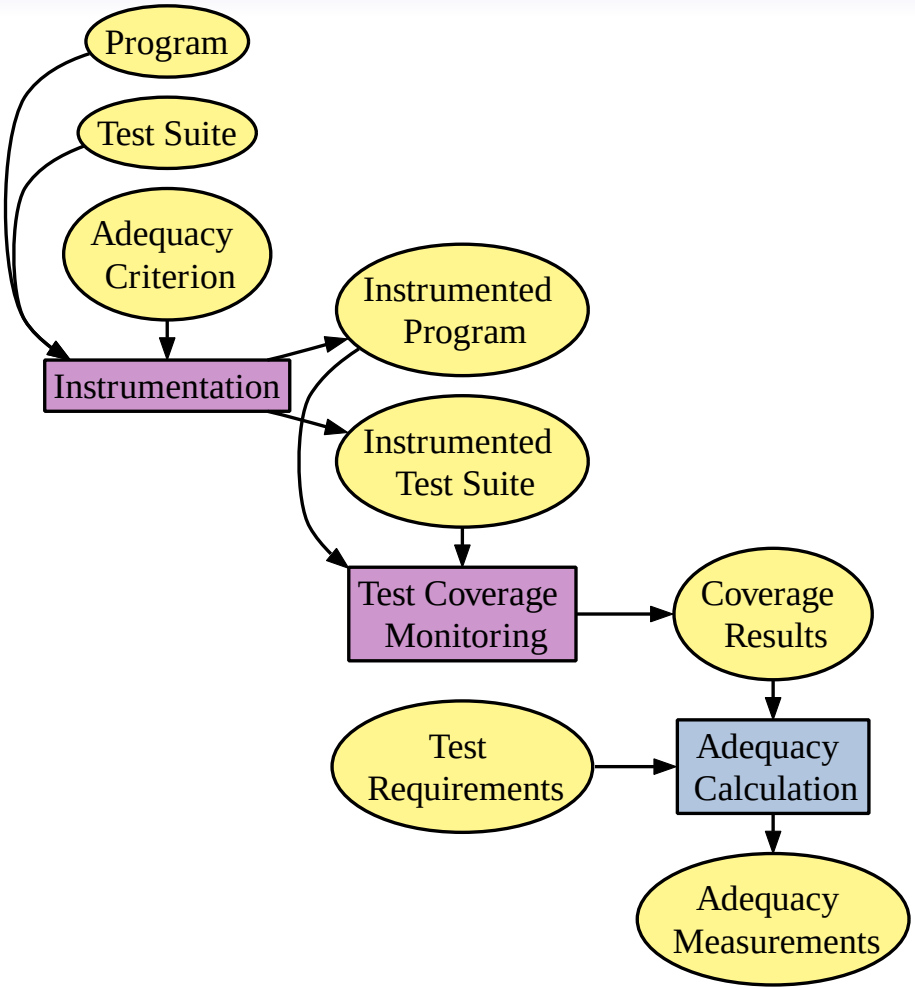
Calculating Coverage

Use instrumentation probes to **capture** and **analyze** a test suite's coverage of the program state and structure

Regression Testing

The coverage results and adequacy measurements can support both test suite **reduction** and **prioritization**

Calculating the Coverage of a Test Suite



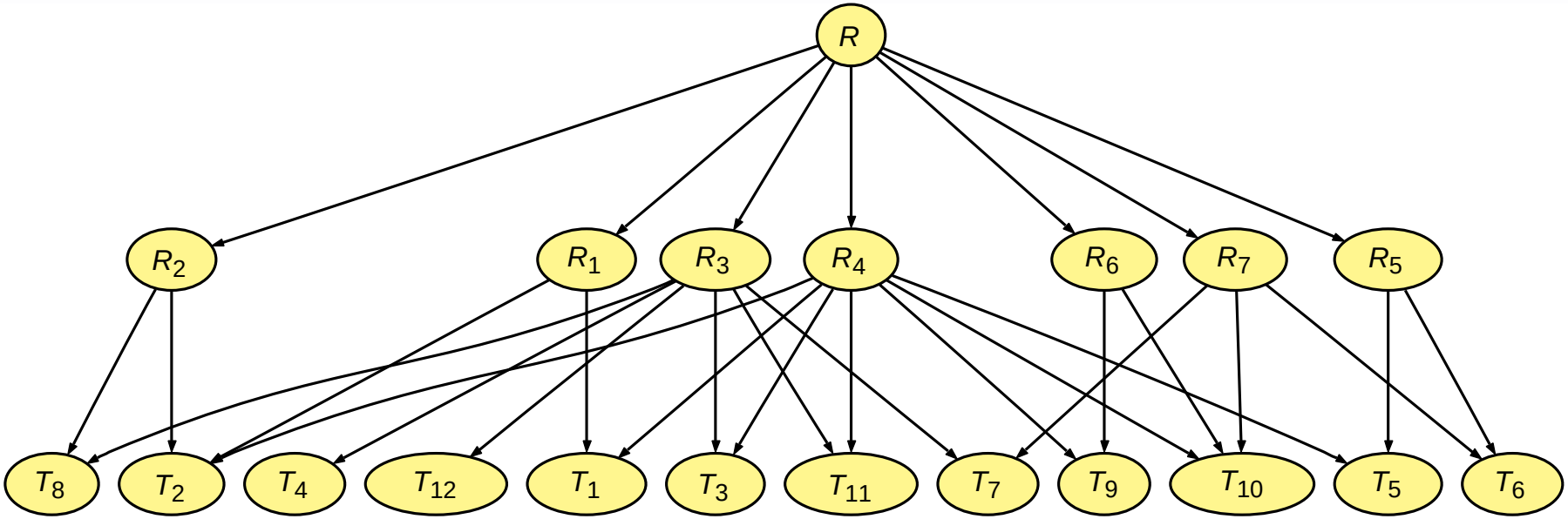
Calculating Coverage

Use instrumentation probes to **capture** and **analyze** a test suite's coverage of the program state and structure

Regression Testing

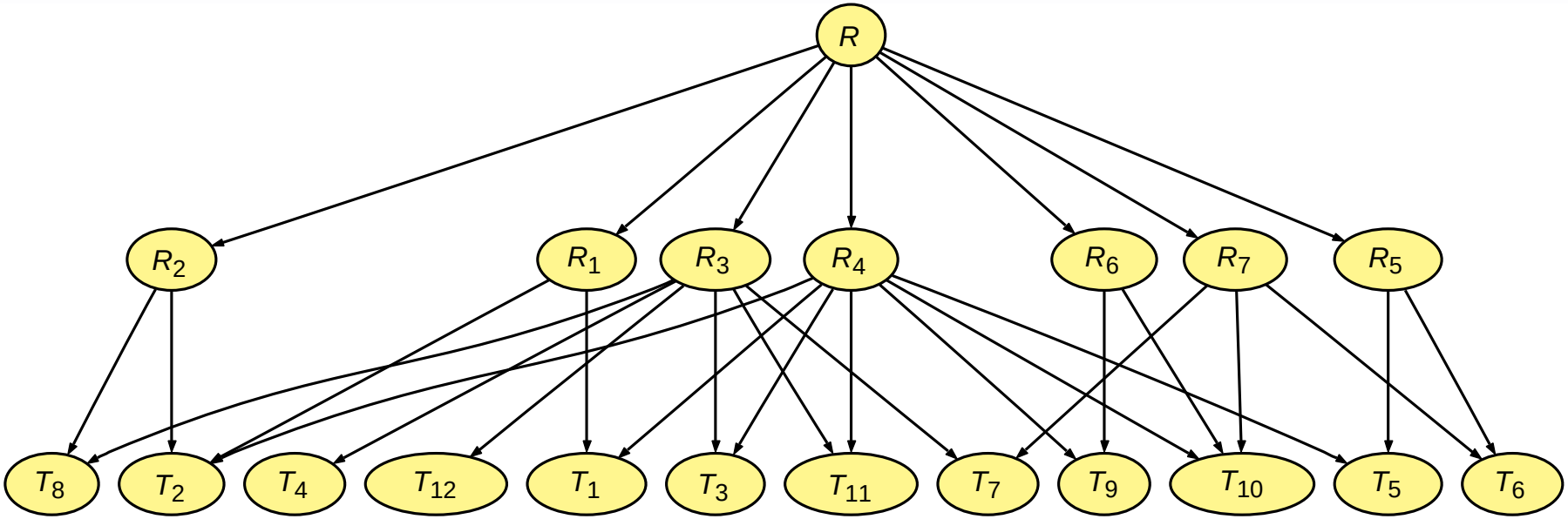
The coverage results and adequacy measurements can support both test suite **reduction** and **prioritization**

Finding the Overlap in Coverage



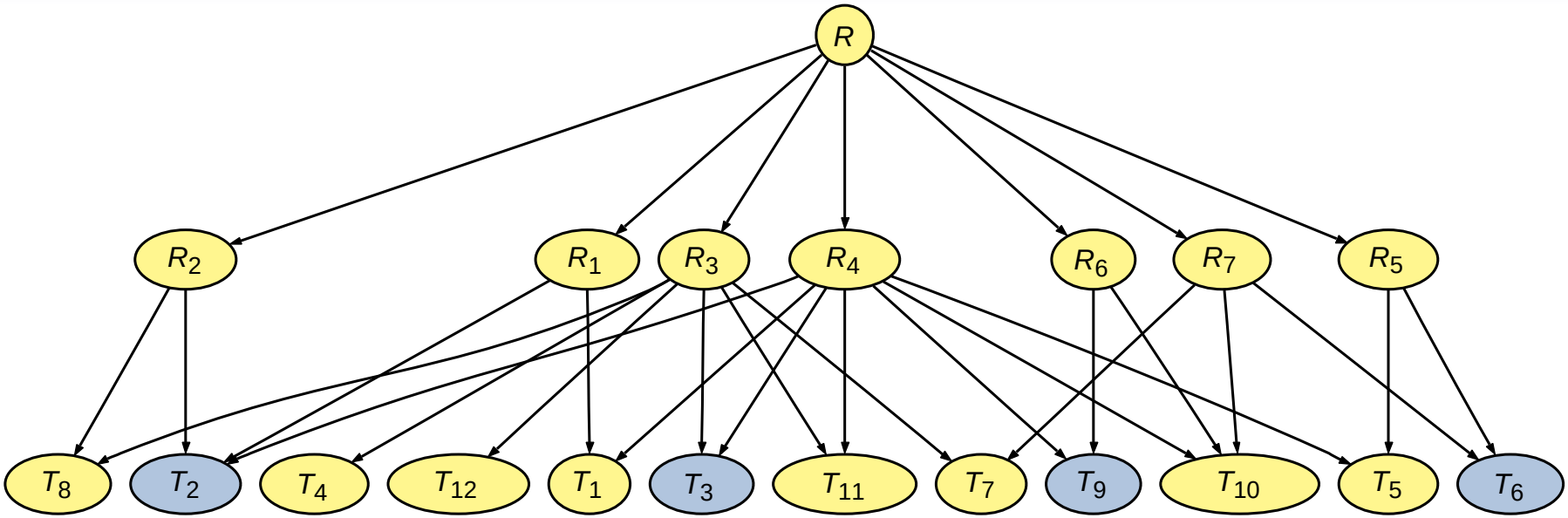
- $R_j \rightarrow T_i$ means that requirement R_j is **covered by** test T_i
- Test suite **reduction** discards the test cases that **redundantly** cover the test requirements
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements

Finding the Overlap in Coverage



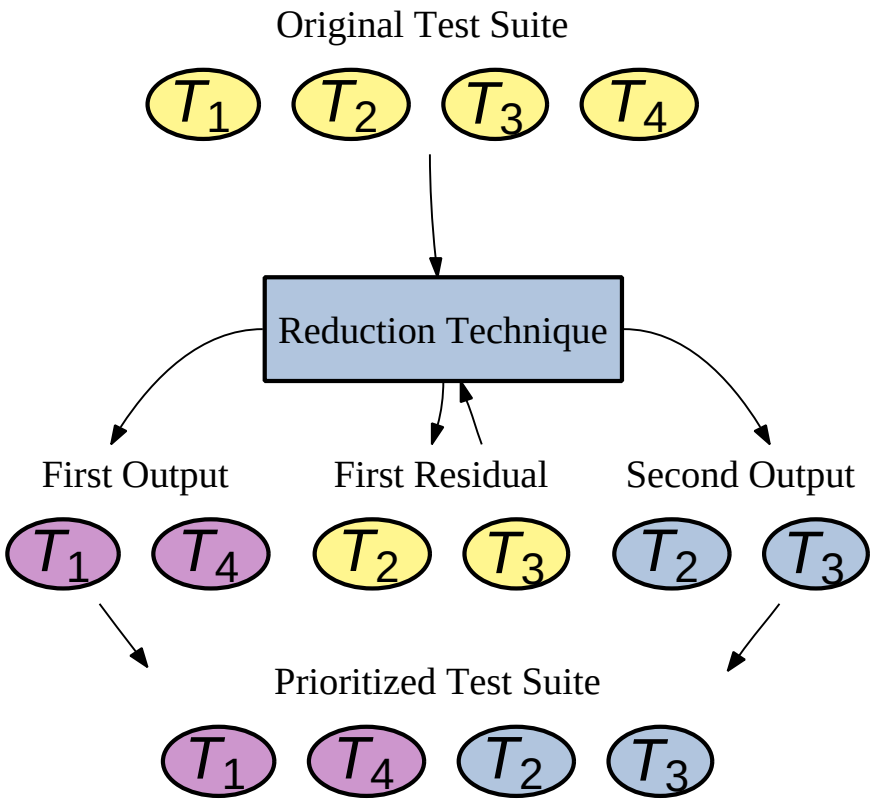
- $R_j \rightarrow T_i$ means that requirement R_j is **covered by** test T_i
- Test suite **reduction** discards the test cases that **redundantly** cover the test requirements
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements

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

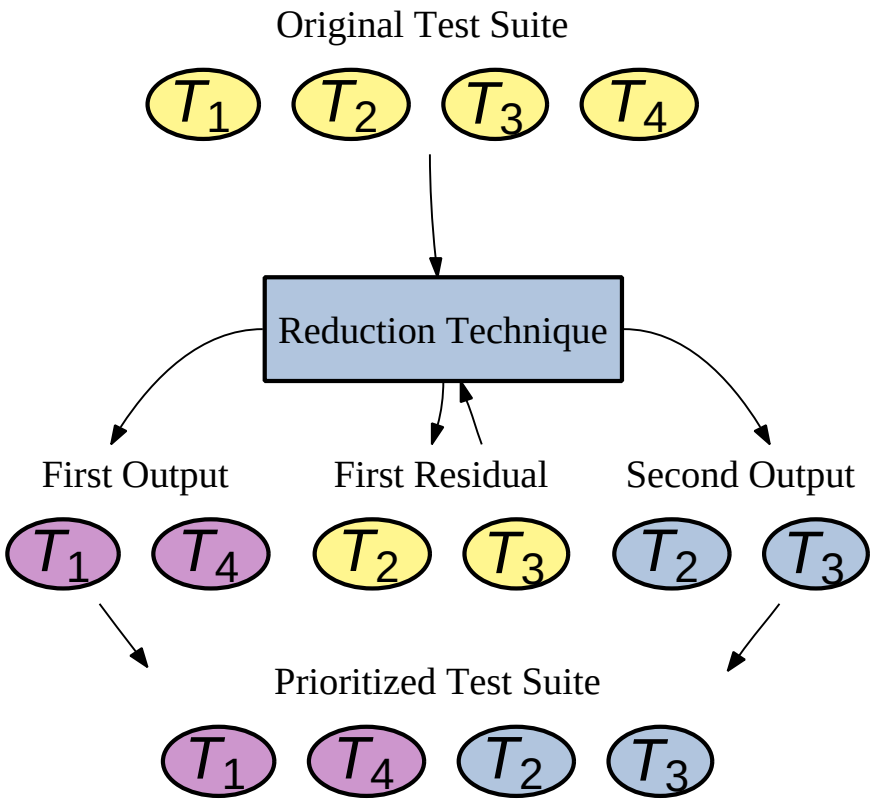


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

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

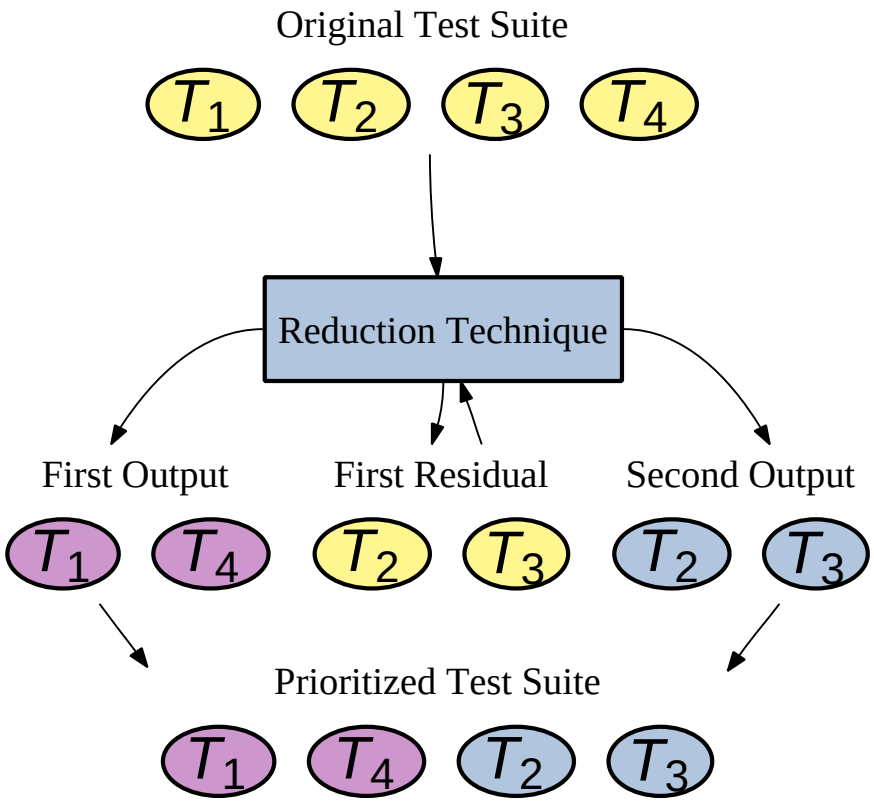


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

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

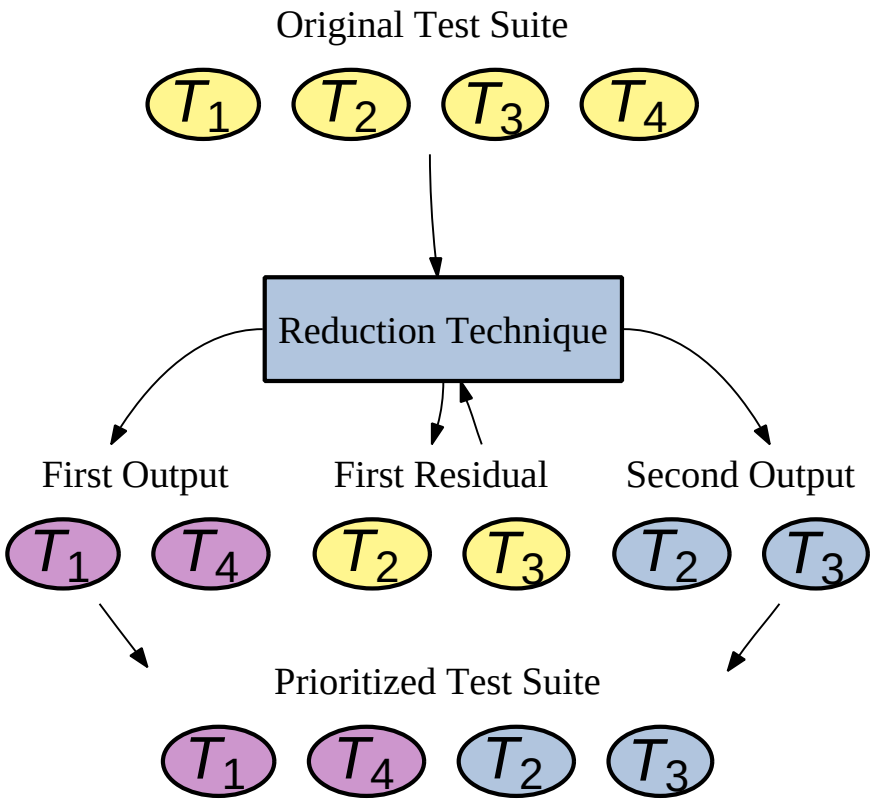


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

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

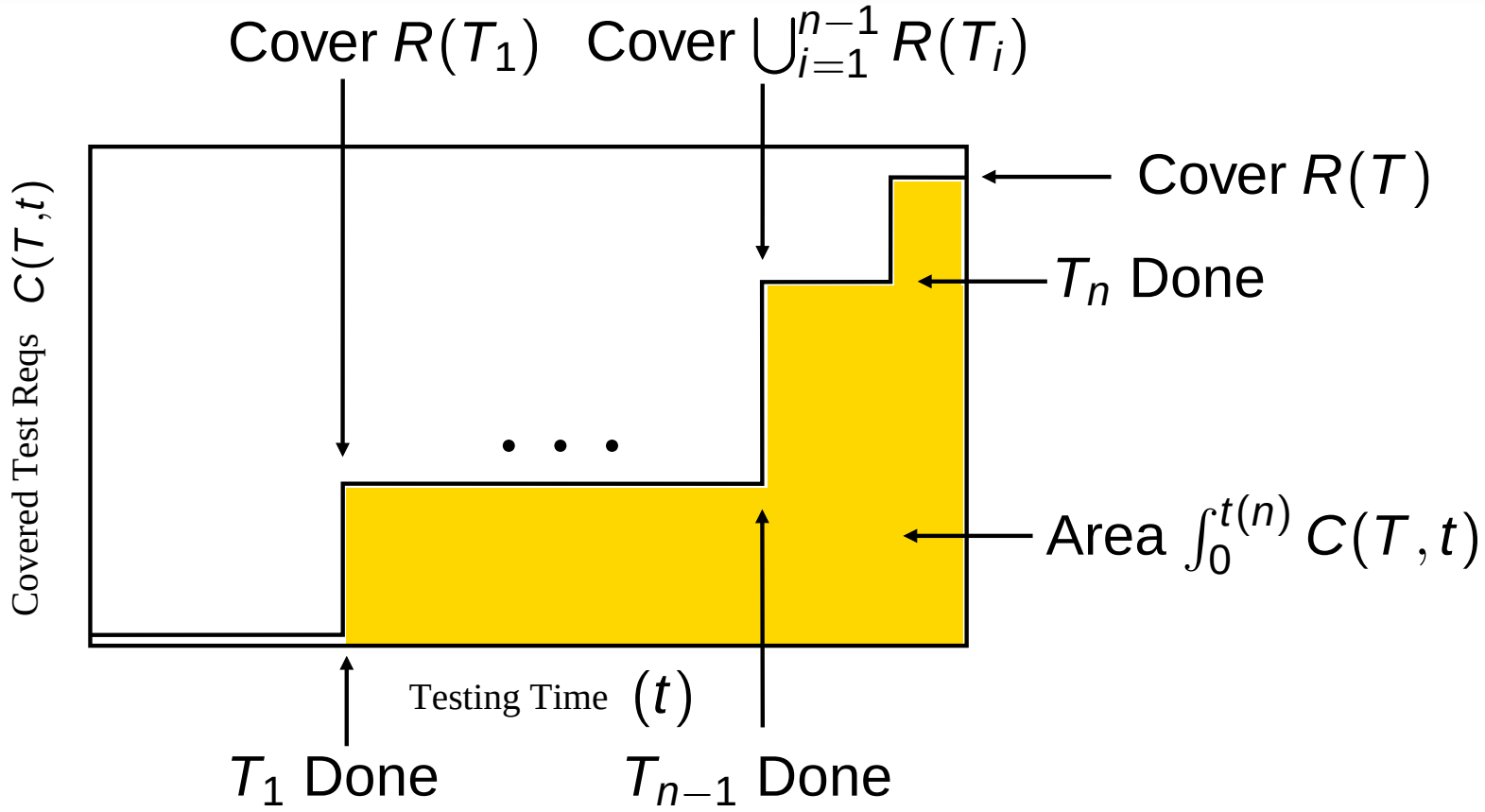


- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

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

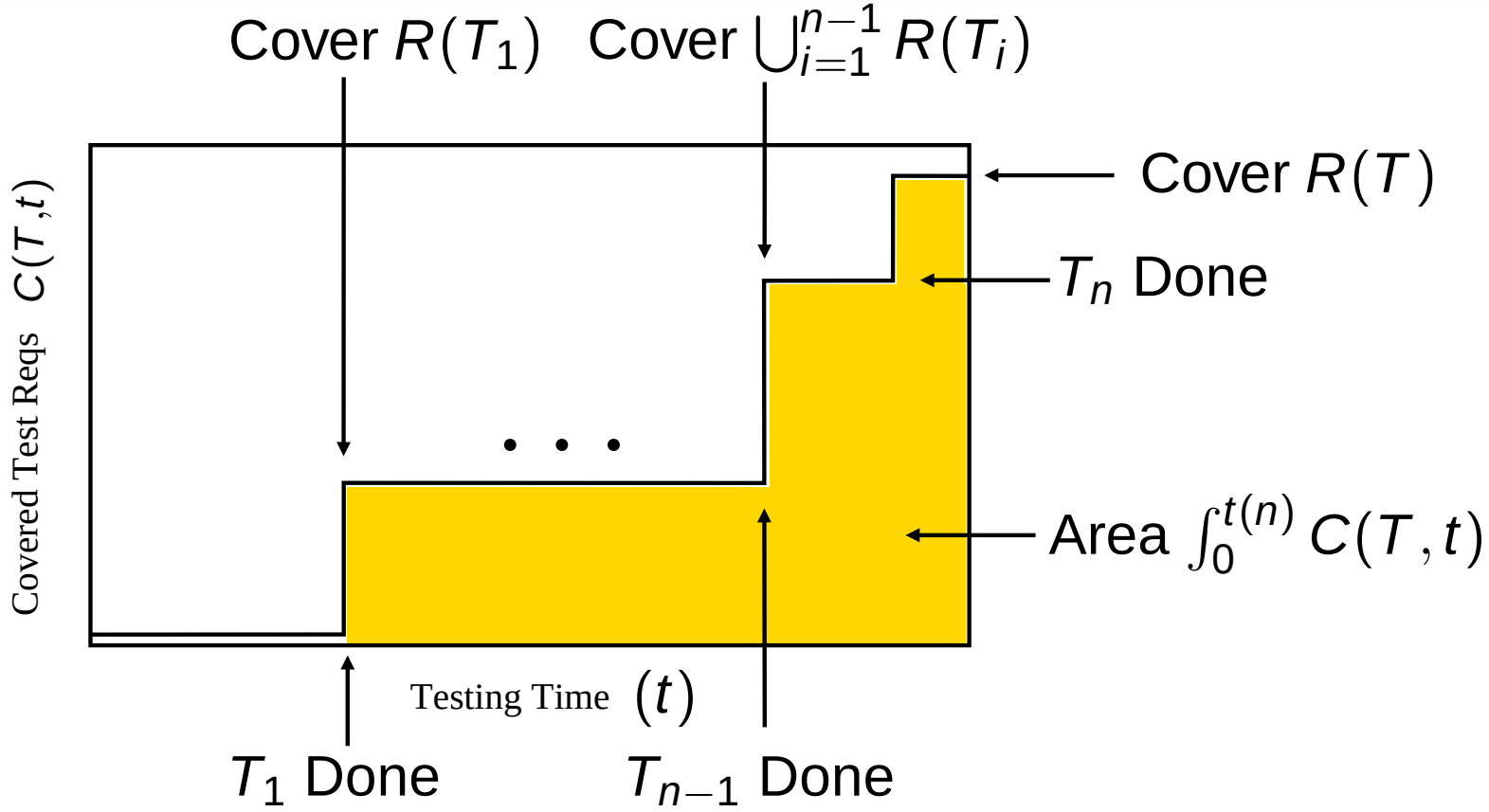
Compare (i) **greedy choices** (cost, coverage, and ratio) and (ii) **algorithms**

Evaluating Test Suite Prioritizers



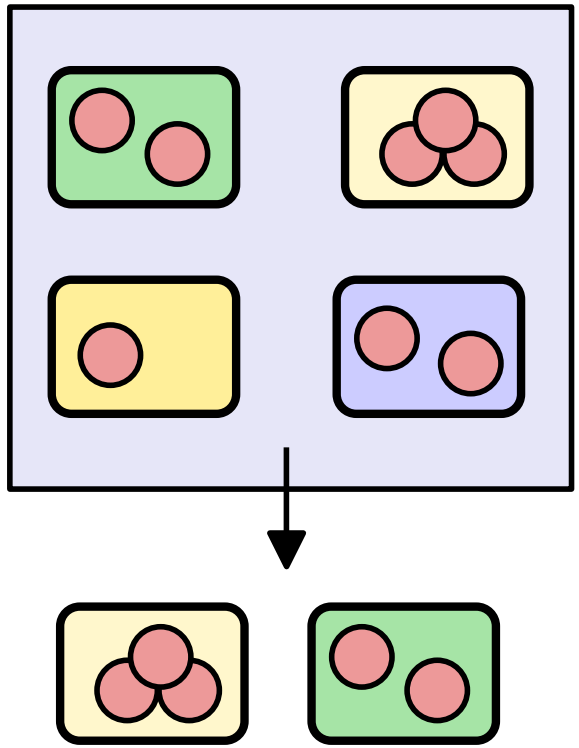
Prioritize to **increase** the CE of a test suite $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0,1]$

Evaluating Test Suite Prioritizers



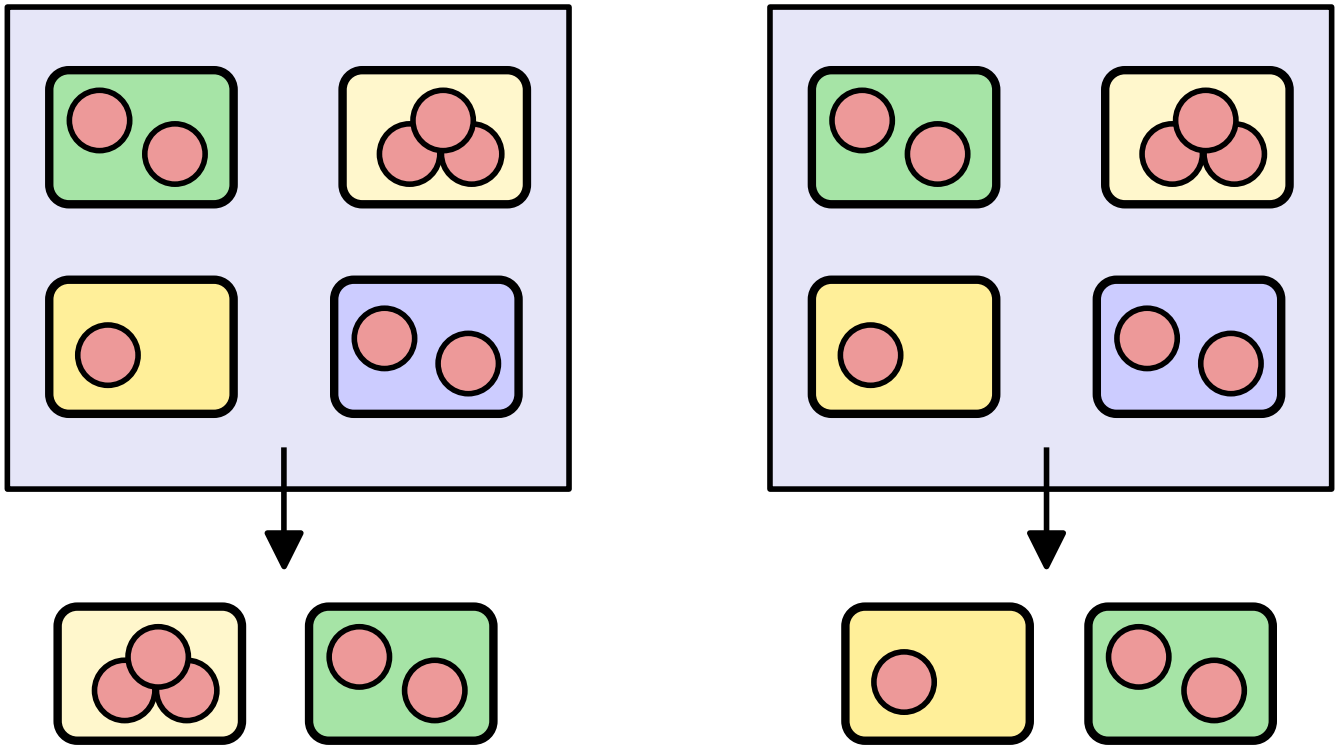
Prioritize to **increase** the CE of a test suite $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0,1]$

Evaluating Test Suite Reducers



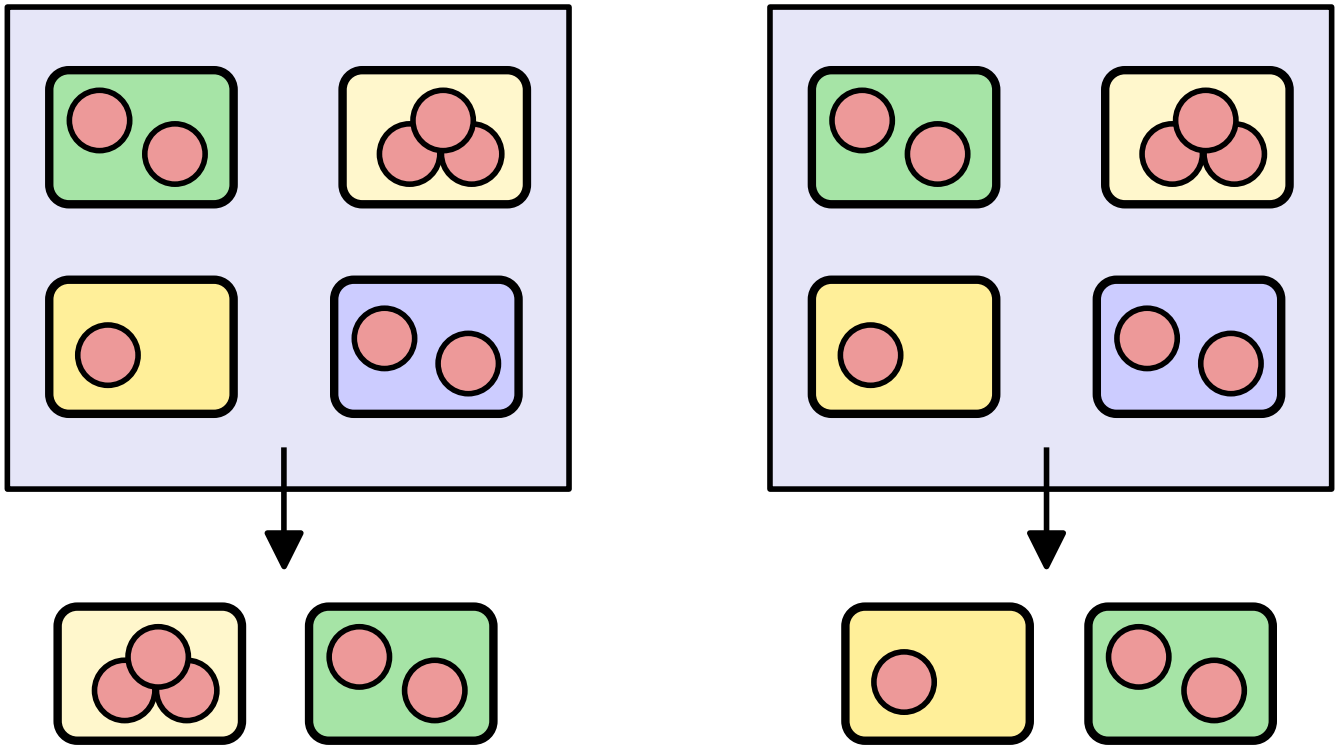
Reduction Factor for Size (RFFS): How small is the reduced test suite?

Evaluating Test Suite Reducers



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

Evaluating Test Suite Reducers



Common Rate (CR): How similar are differently reduced test suites?

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

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

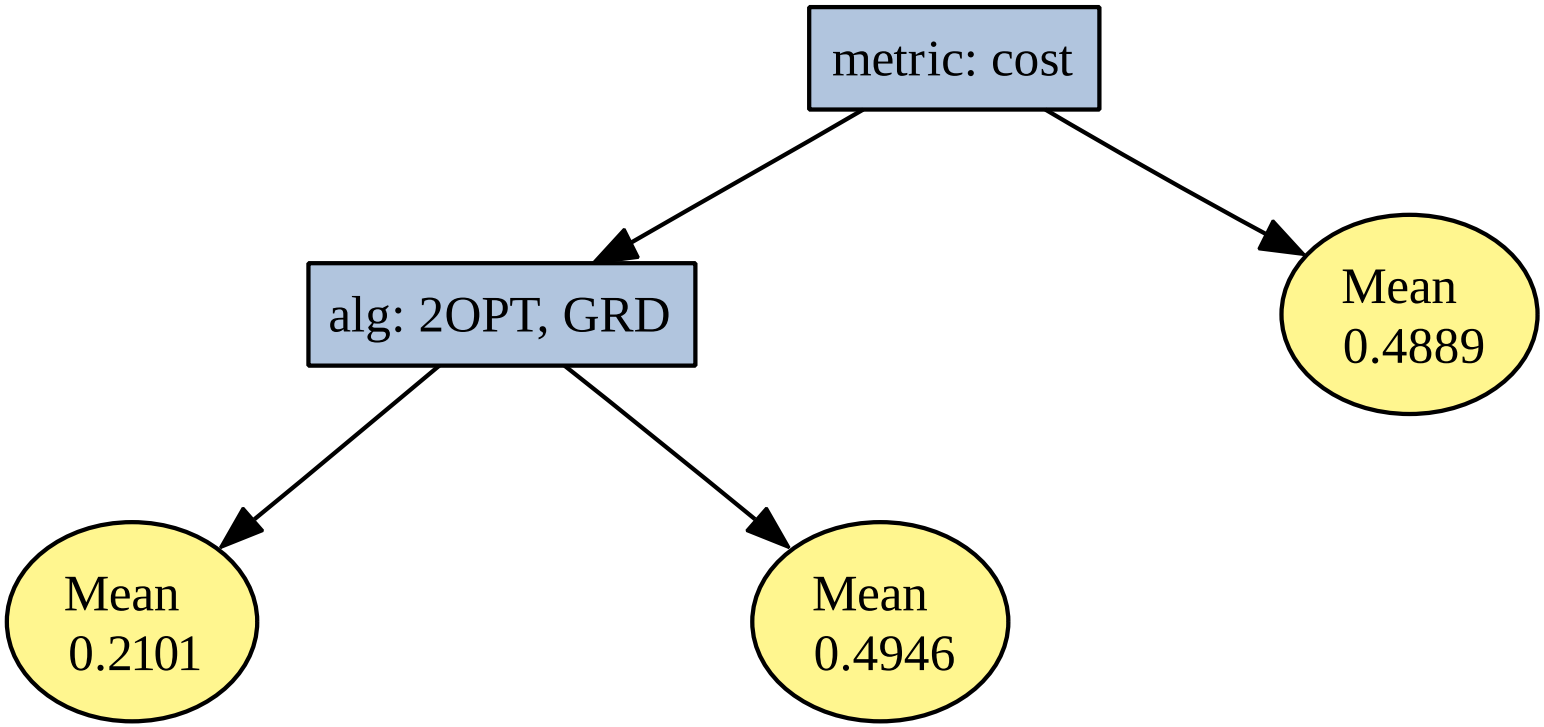
Greedy-by	T_r	$time(T_r)$	T_p	CE
coverage	$\langle T_1, T_4 \rangle$	5	$\langle T_1, T_4, T_2, T_3 \rangle$	0.400
time	$\langle T_2, T_3, T_4 \rangle$	3	$\langle T_2, T_3, T_4, T_1 \rangle$	0.714
ratio	$\langle T_2, T_4, T_3 \rangle$	3	$\langle T_2, T_4, T_3, T_1 \rangle$	0.743

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

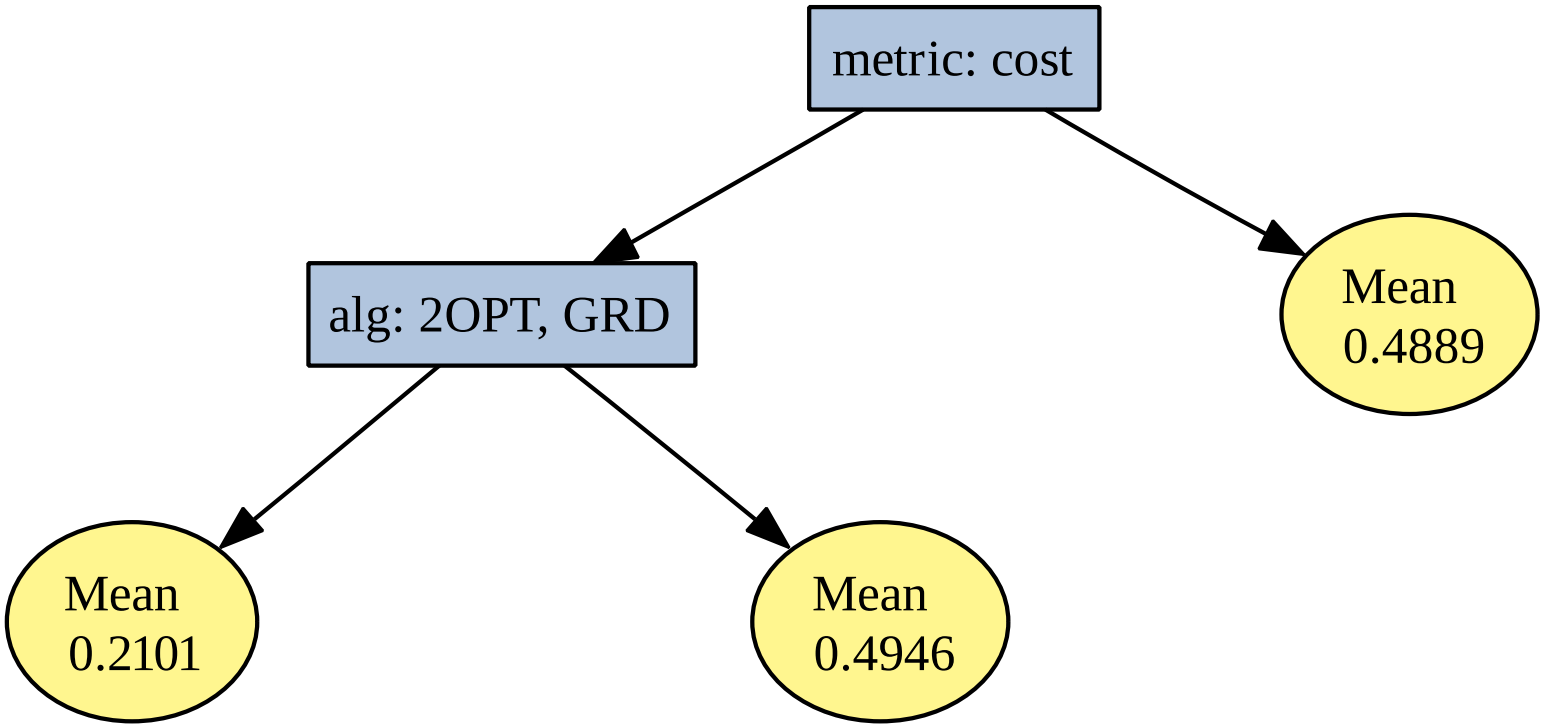
Greedy-by	T_r	$time(T_r)$	T_p	CE
coverage	$\langle T_1, T_4 \rangle$	5	$\langle T_1, T_4, T_2, T_3 \rangle$	0.400
time	$\langle T_2, T_3, T_4 \rangle$	3	$\langle T_2, T_3, T_4, T_1 \rangle$	0.714
ratio	$\langle T_2, T_4, T_3 \rangle$	3	$\langle T_2, T_4, T_3, T_1 \rangle$	0.743

Analysis Method: Tree Models



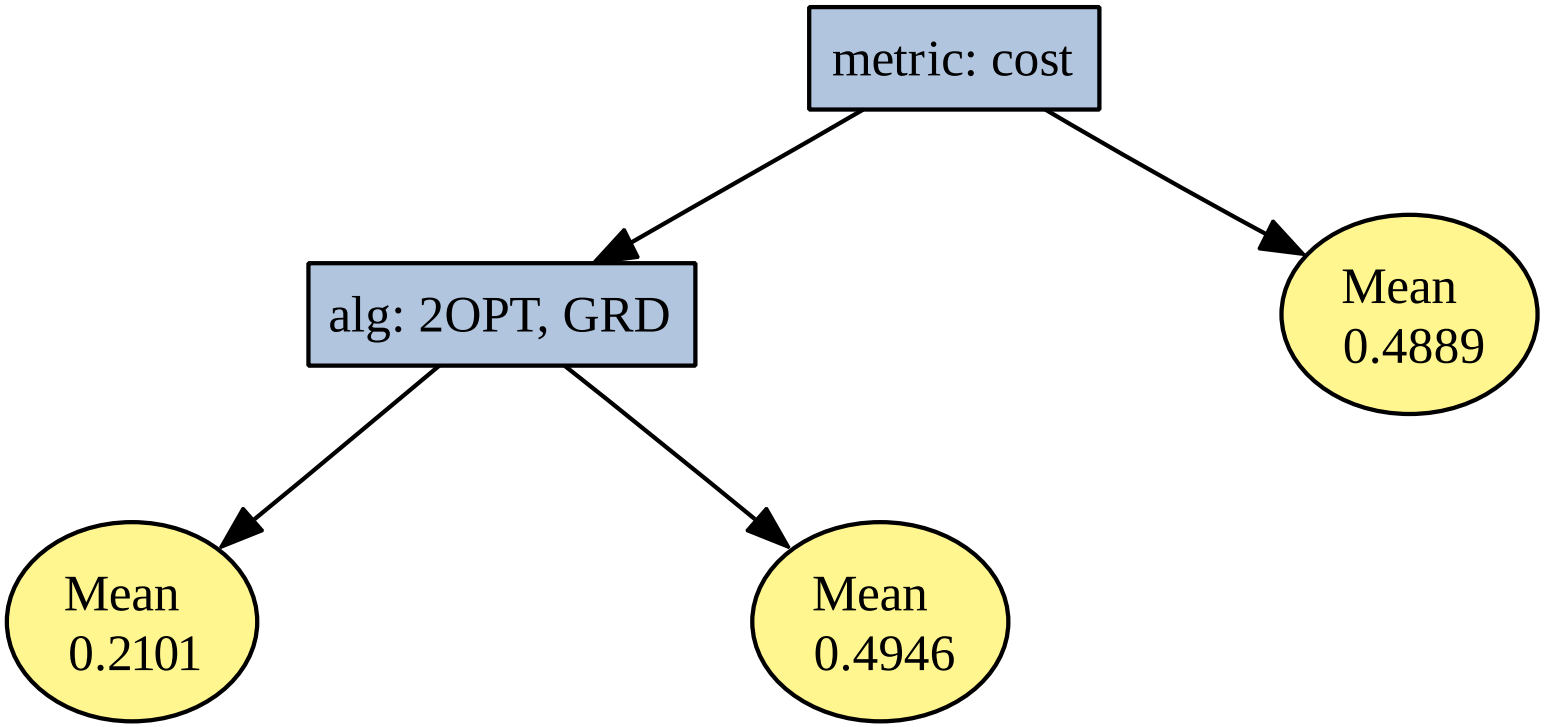
Tree Models: Use recursive partitioning to create hierarchical view of data

Analysis Method: Tree Models



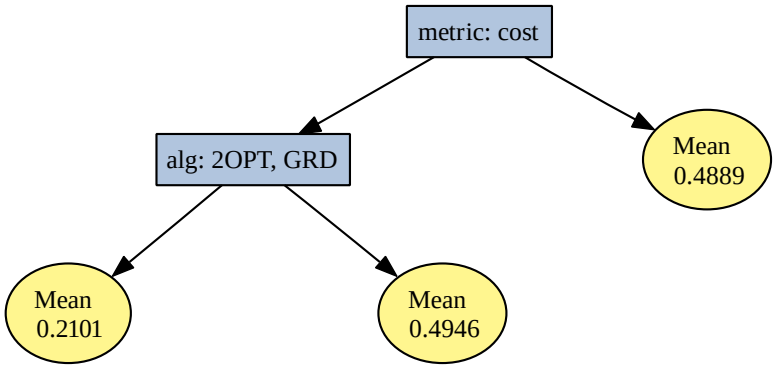
Explanatory Variable: Configuration of the testing methods (e.g., GCM)

Analysis Method: Tree Models

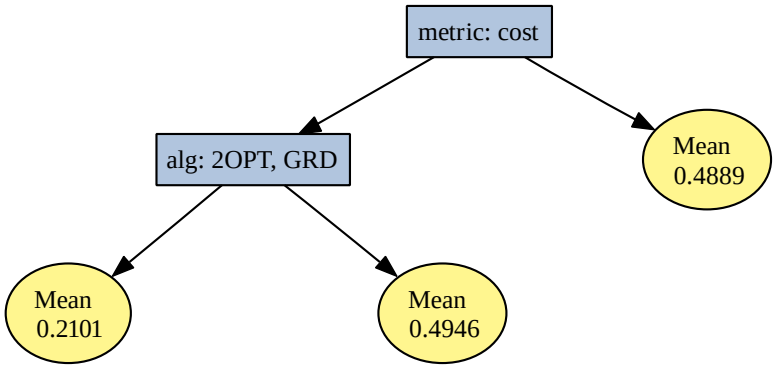


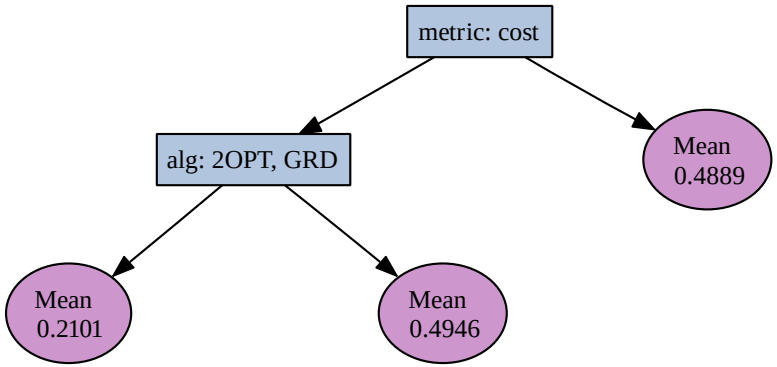
Response Variable: One of the evaluation metrics (e.g., CE or RFFT)

Analysis Method: Random Forests

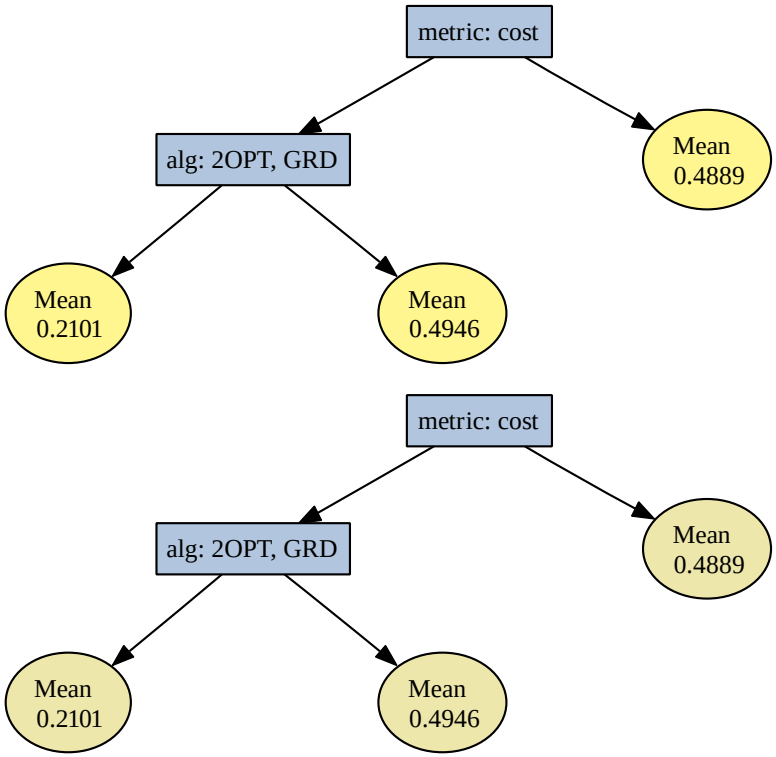


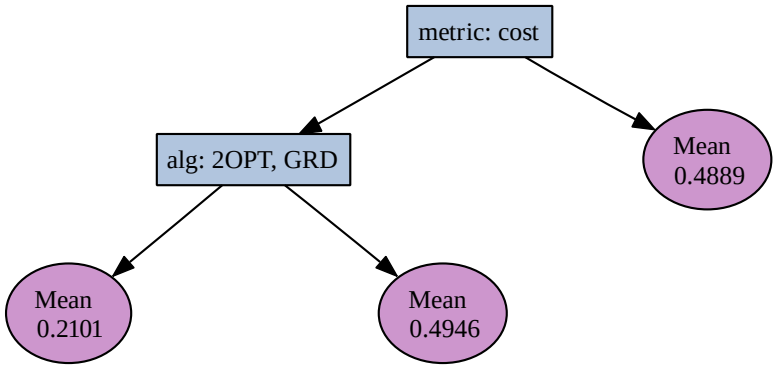
Analysis Method: Random Forests



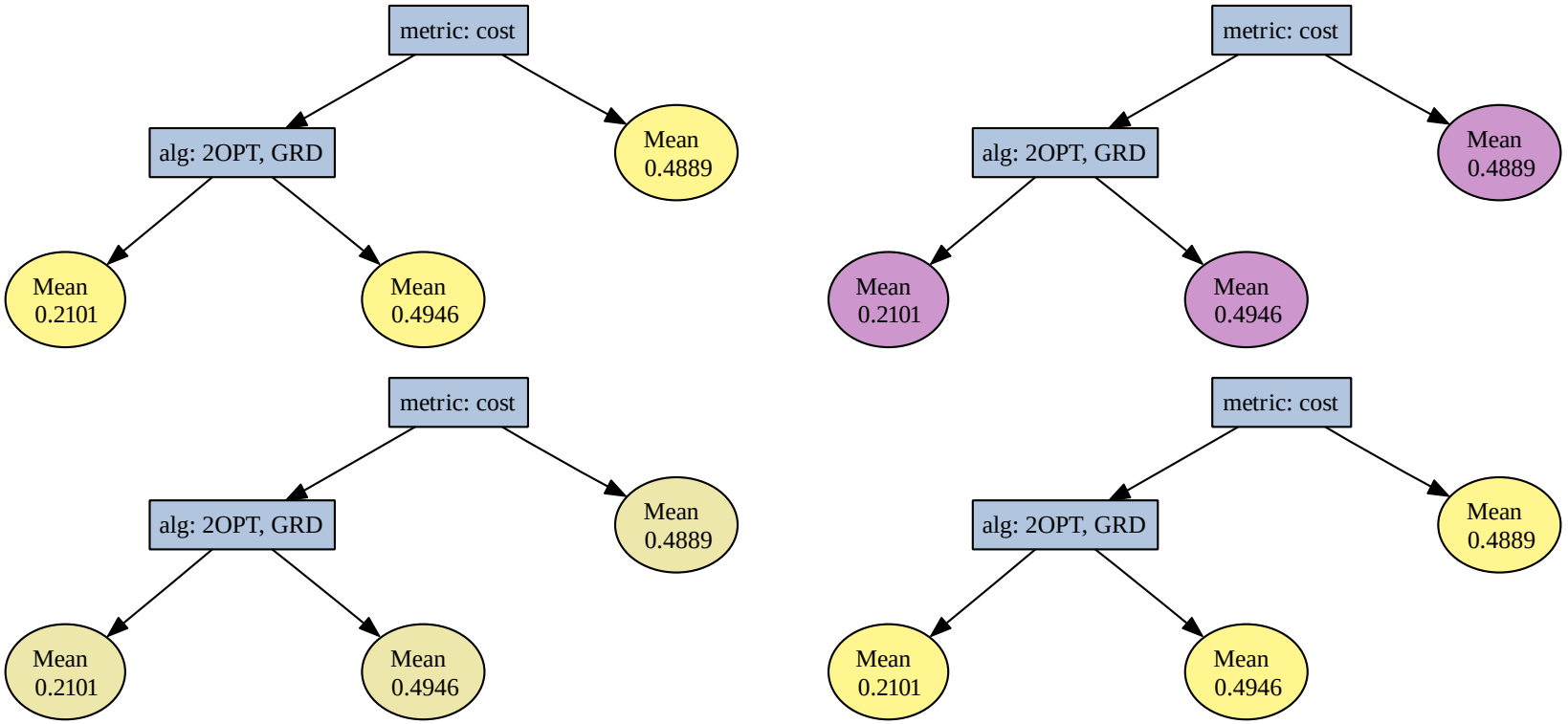


Analysis Method: Random Forests

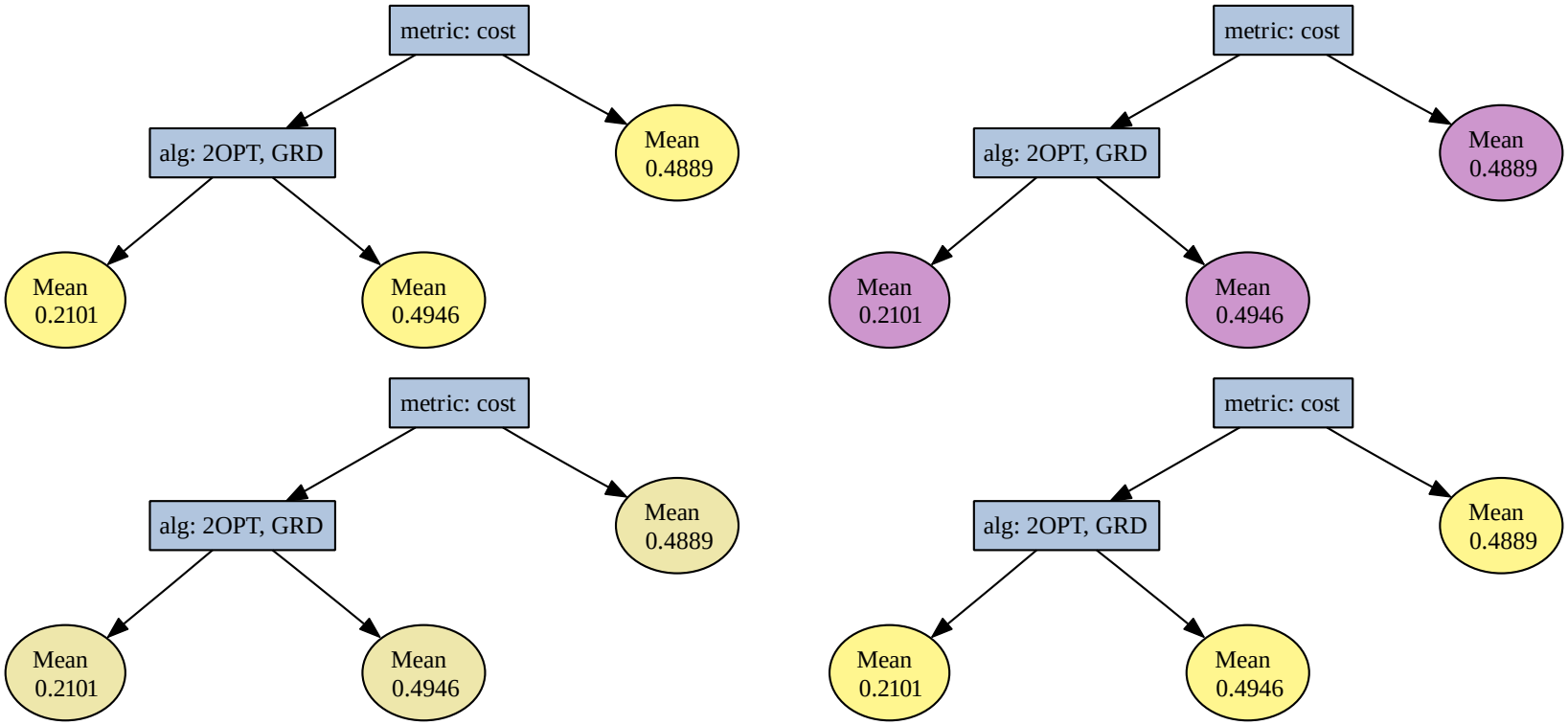




Analysis Method: Random Forests



Analysis Method: Random Forests



Many Trees: Randomly construct a large collection of trees in order to avoid bias and identify the most important explanatory variables

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	$\|T\|$	$\|\mathcal{R}(T)\|$	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

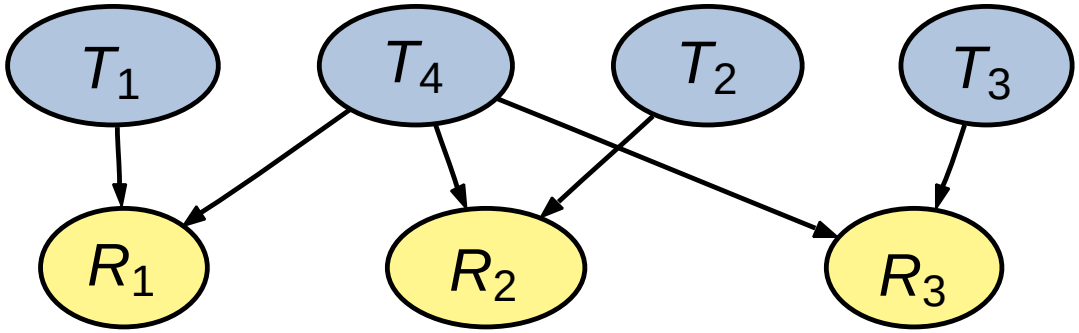
Name	$\|T\|$	$\|\mathcal{R}(T)\|$	CCN	NCSS
DS	110	40	1.35	1243.00
GB	51	88	2.60	1455.00
JD	54	783	1.64	2716.00
LF	13	6	1.40	215.00
RM	13	19	2.13	569.00
SK	27	117	2.00	628.00
TM	27	46	2.21	748.00
RP	76	221	2.65	6822.00

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

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

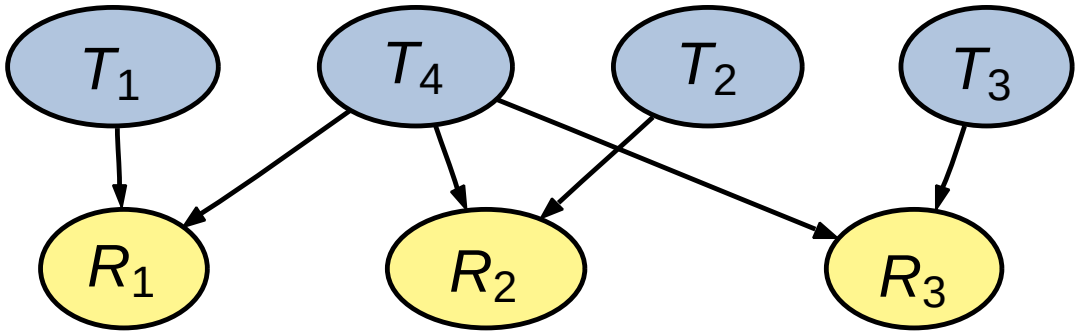
"Greedy Fooling" Coverage Generation



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 $T_p = \langle T_n, \ldots, T_1 \rangle$ even though $CE(T_p) < CE(T)$ for $T = \langle T_1, \ldots, T_n \rangle$
- **Inspiration**: Vazirani's construction of a **tight example** for the greedy **minimal set cover** algorithm

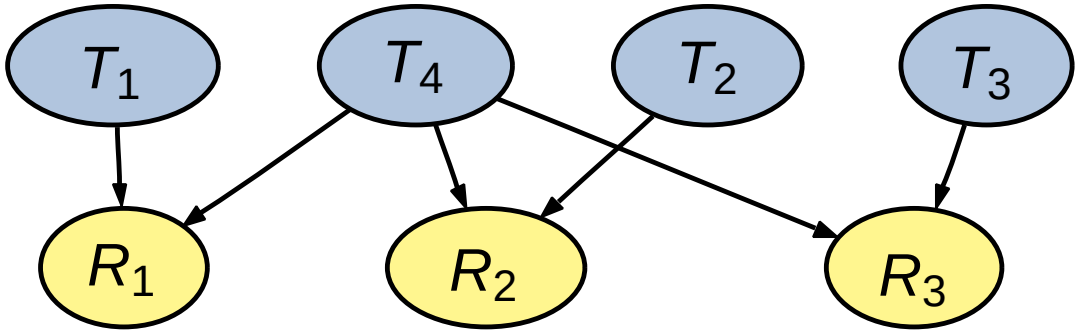
"Greedy Fooling" Coverage Generation



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 $T_p = \langle T_n, \ldots, T_1 \rangle$ even though $CE(T_p) < CE(T)$ for $T = \langle T_1, \ldots, T_n \rangle$
- **Inspiration**: Vazirani's construction of a **tight example** for the greedy **minimal set cover** algorithm

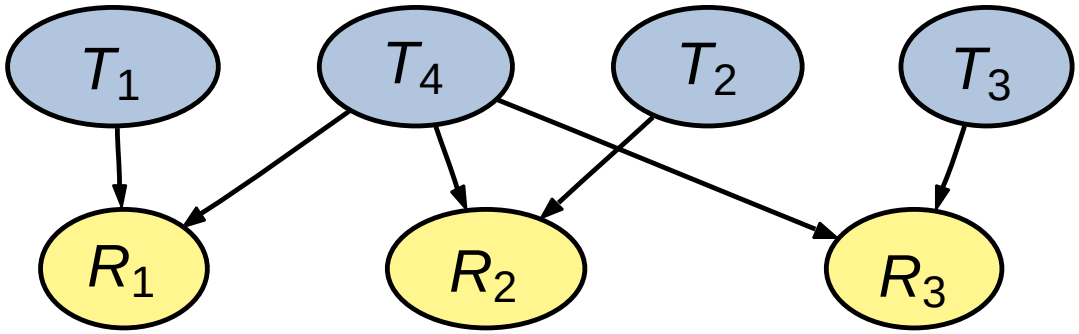
"Greedy Fooling" Coverage Generation



Generation Procedure

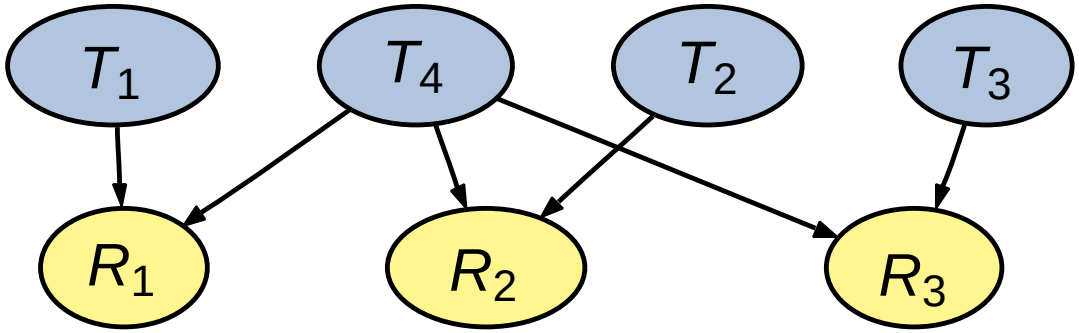
- The **greedy** test prioritizer iteratively selects test cases according to the **(coverage / cost)** ratio
- **Goal**: generate coverage and timing information that will **fool** the greedy technique into creating $T_p = \langle T_n, \ldots, T_1 \rangle$ even though $CE(T_p) < CE(T)$ for $T = \langle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

Constructing "Greedy Fooling" Test Suites



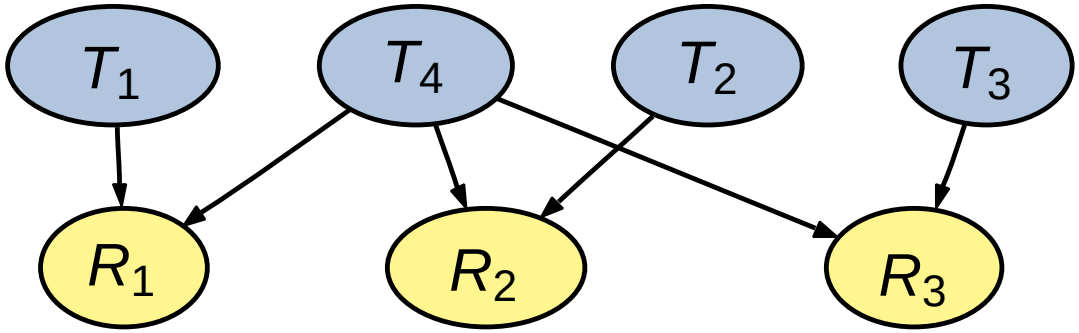
- **Approach**: use one dimensional **optimization** (e.g., golden section search and successive parabolic interpolation) to pick a value for $cost(T_n)$
- **Construction**: set $cost(T_1) = cost(T_2) = cost(T_3) = 1$ and then determine the bounds for $cost(T_4) \in [C_{min}, C_{max}]$
- **Example**: $cost(T_4) \in [2.138803, 2.472136]$ so that
  $CE_{min}(T_p) = .5838004 \quad CE_{min}(T) = .6108033$
  $CE_{max}(T_p) = .5482172 \quad CE_{max}(T) = .6345125$

Constructing "Greedy Fooling" Test Suites



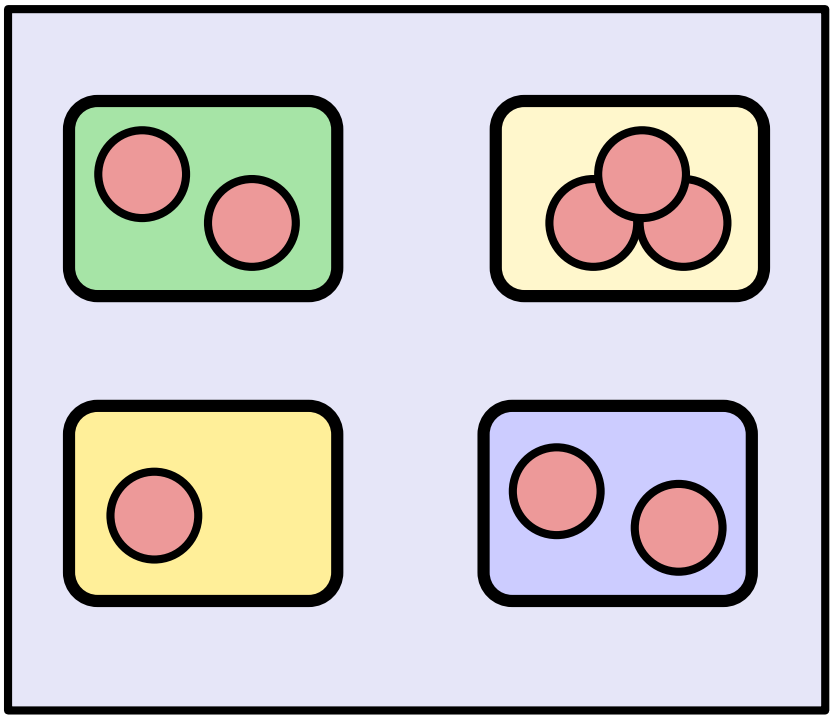
- **Approach**: use one dimensional **optimization** (e.g., golden section search and successive parabolic interpolation) to pick a value for $cost(T_n)$
- **Construction**: set $cost(T_1) = cost(T_2) = cost(T_3) = 1$ and then determine the bounds for $cost(T_4) \in [C_{min}, C_{max}]$
- **Example**: $cost(T_4) \in [2.138803, 2.472136]$ so that
  $CE_{min}(T_p) = .5838004 \quad CE_{min}(T) = .6108033$
  $CE_{max}(T_p) = .5482172 \quad CE_{max}(T) = .6345125$

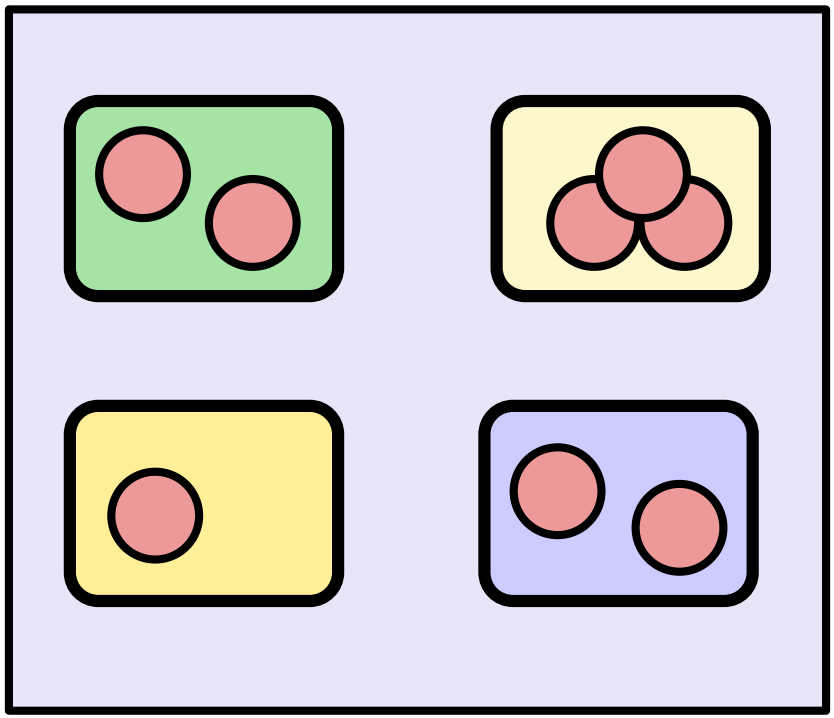
Constructing "Greedy Fooling" Test Suites



- **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
  $CE_{min}(T_p) = .5838004 \quad CE_{min}(T) = .6108033$
  $CE_{max}(T_p) = .5482172 \quad CE_{max}(T) = .6345125$

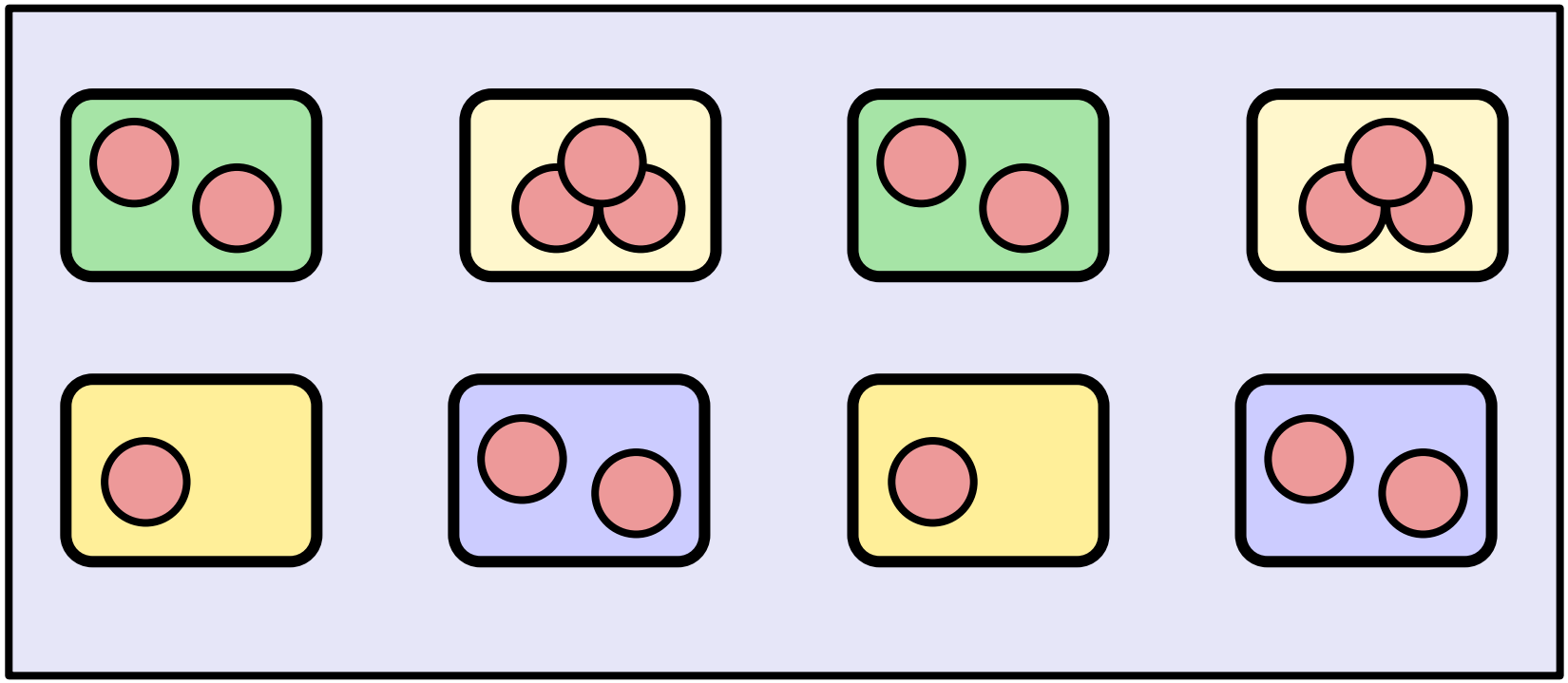
Constructing Build/Test Machine Suites





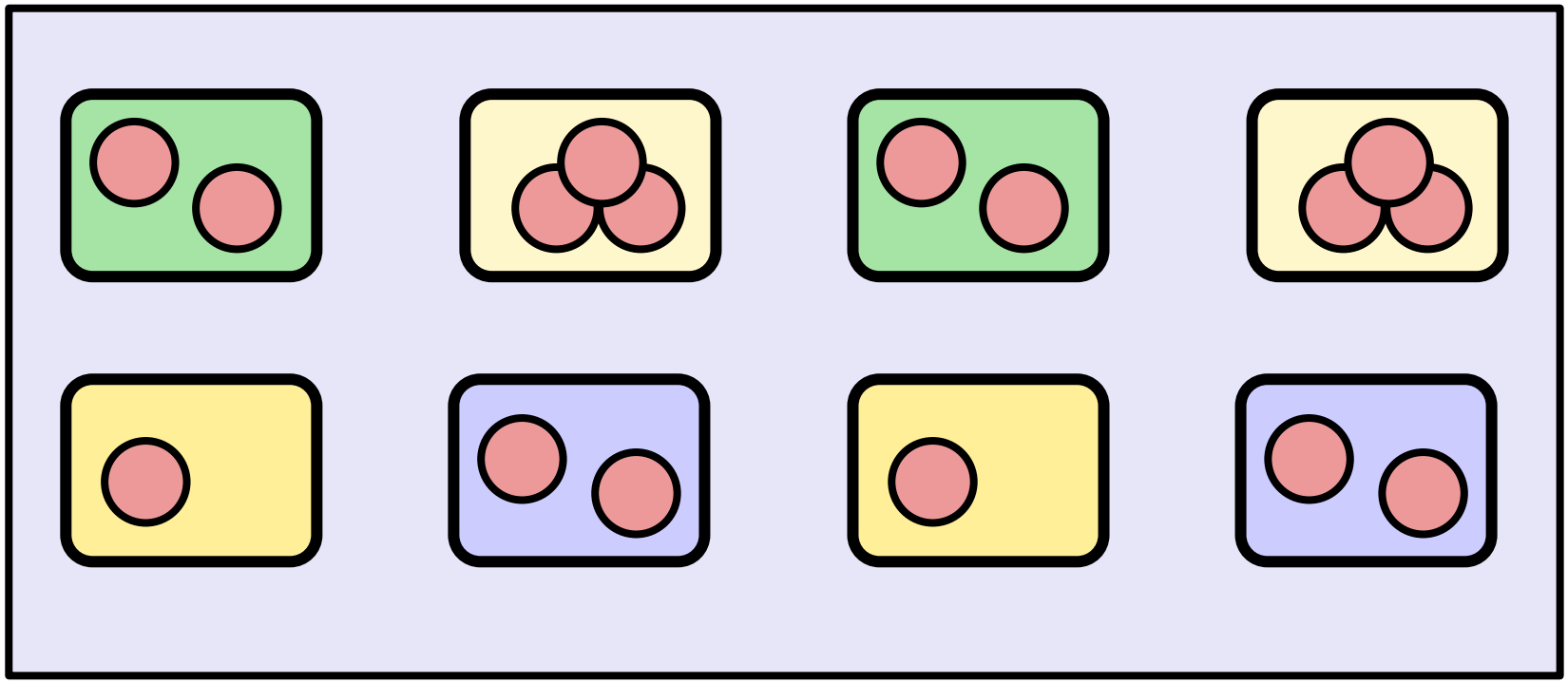
Objective: Simulate test suite execution on a centralized server

Constructing Build/Test Machine Suites



Objective: Simulate test suite execution on a centralized server

Constructing Build/Test Machine Suites



Construction: Combine all of the test suites and coverage reports

Random Test Suite Prioritization

Random Number	Input	Output
$\{1,2,3,4\}$ 2	$\langle t_1, t_2, t_3, \lvert t_4 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2,3\}$ 3	$\langle t_1, t_4, \lvert t_3, t_2 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2\}$ 1	$\langle t_1, \lvert t_4, t_3, t_2 \rangle$	$\langle t_4, t_1, t_3, t_2 \rangle$

Random Test Suite Prioritization

Random Number	Input	Output
$\{1,2,3,4\}$ 2	$\langle t_1, t_2, t_3, \vert t_4 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2,3\}$ 3	$\langle t_1, t_4, \vert t_3, t_2 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2\}$ 1	$\langle t_1, \vert t_4, t_3, t_2 \rangle$	$\langle t_4, t_1, t_3, t_2 \rangle$

Random Test Suite Prioritization

Random Number	Input	Output
$\{1,2,3,4\}$ 2	$\langle t_1, t_2, t_3, \vert t_4 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2,3\}$ 3	$\langle t_1, t_4, \vert t_3, t_2 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2\}$ 1	$\langle t_1, \vert t_4, t_3, t_2 \rangle$	$\langle t_4, t_1, t_3, t_2 \rangle$

Random Test Suite Prioritization

Random Number	Input	Output
$\{1,2,3,4\}$ 2	$\langle t_1, t_2, t_3, \vert t_4 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2,3\}$ 3	$\langle t_1, t_4, \vert t_3, t_2 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2\}$ 1	$\langle t_1, \vert t_4, t_3, t_2 \rangle$	$\langle t_4, t_1, t_3, t_2 \rangle$

Random Test Suite Prioritization

Random Number	Input	Output
$\{1,2,3,4\}$ 2	$\langle t_1, t_2, t_3, \vert t_4 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2,3\}$ 3	$\langle t_1, t_4, \vert t_3, t_2 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2\}$ 1	$\langle t_1, \vert t_4, t_3, t_2 \rangle$	$\langle t_4, t_1, t_3, t_2 \rangle$

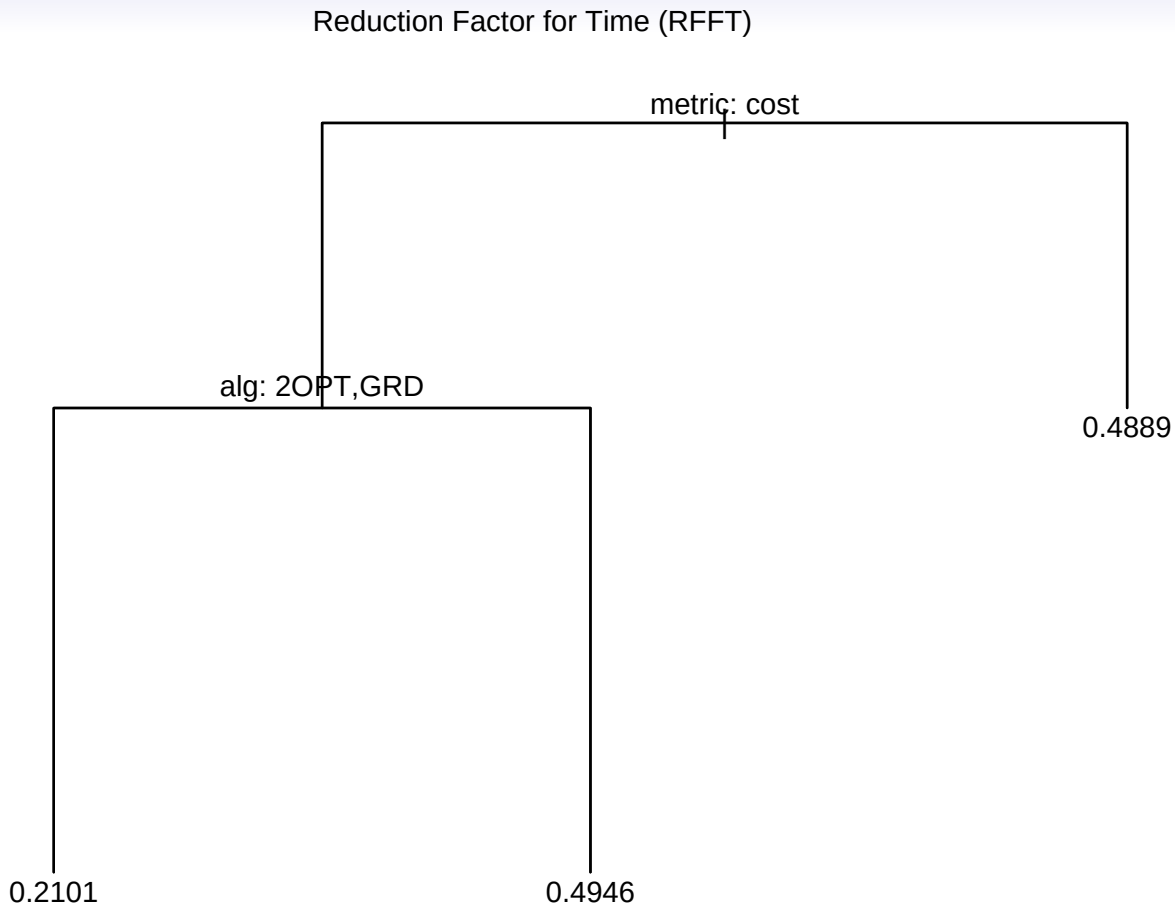
Importance: Random prioritization serves as a valuable experimental control and often produces orderings better than the initial suite

Random Test Suite Prioritization

Random Number	Input	Output
$\{1,2,3,4\}$ 2	$\langle t_1, t_2, t_3, \vert t_4 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2,3\}$ 3	$\langle t_1, t_4, \vert t_3, t_2 \rangle$	$\langle t_1, t_4, t_3, t_2 \rangle$
$\{1,2\}$ 1	$\langle t_1, \vert t_4, t_3, t_2 \rangle$	$\langle t_4, t_1, t_3, t_2 \rangle$

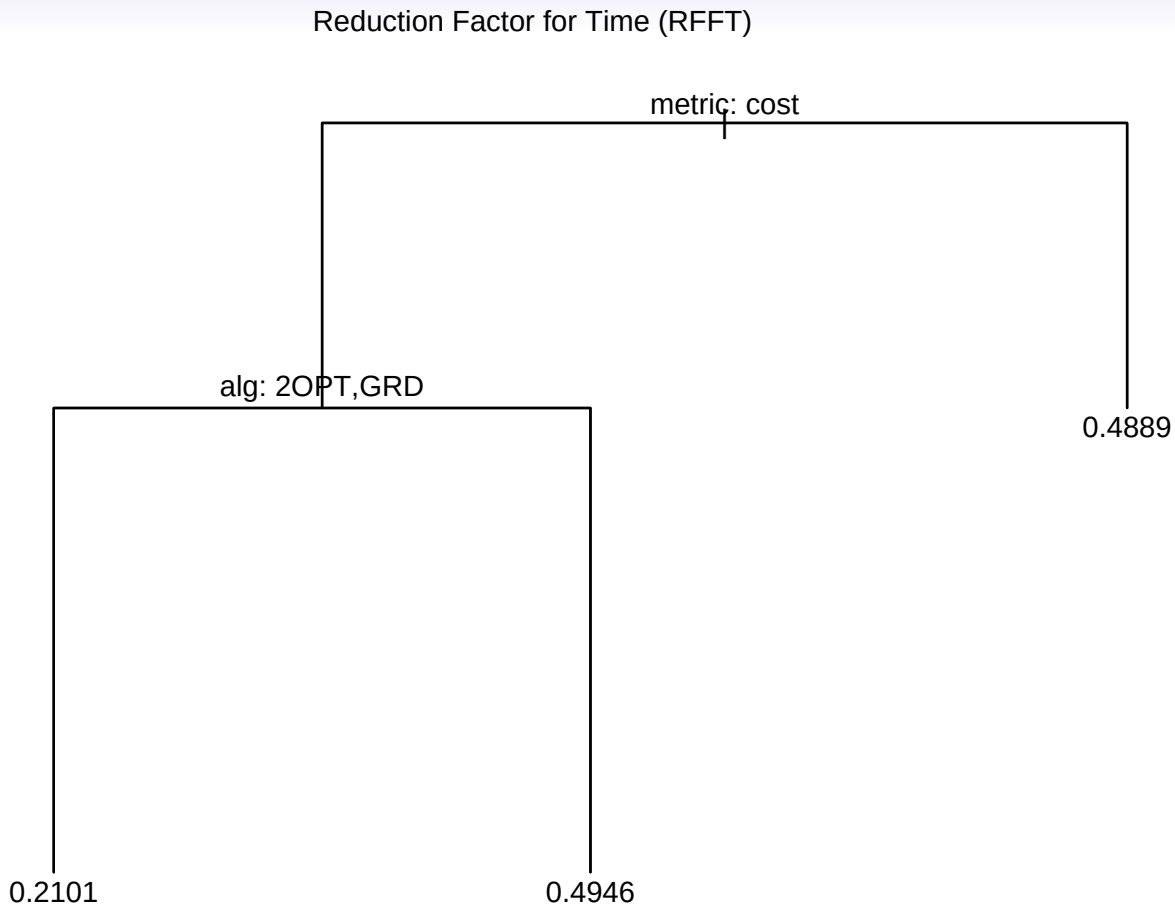
Strategy: Use the modern and efficient implementation of the Fisher-Yates shuffle to produce the reordered test suite $T_p = \langle t_4, t_1, t_3, t_2 \rangle$

Overview of RFFT Trends



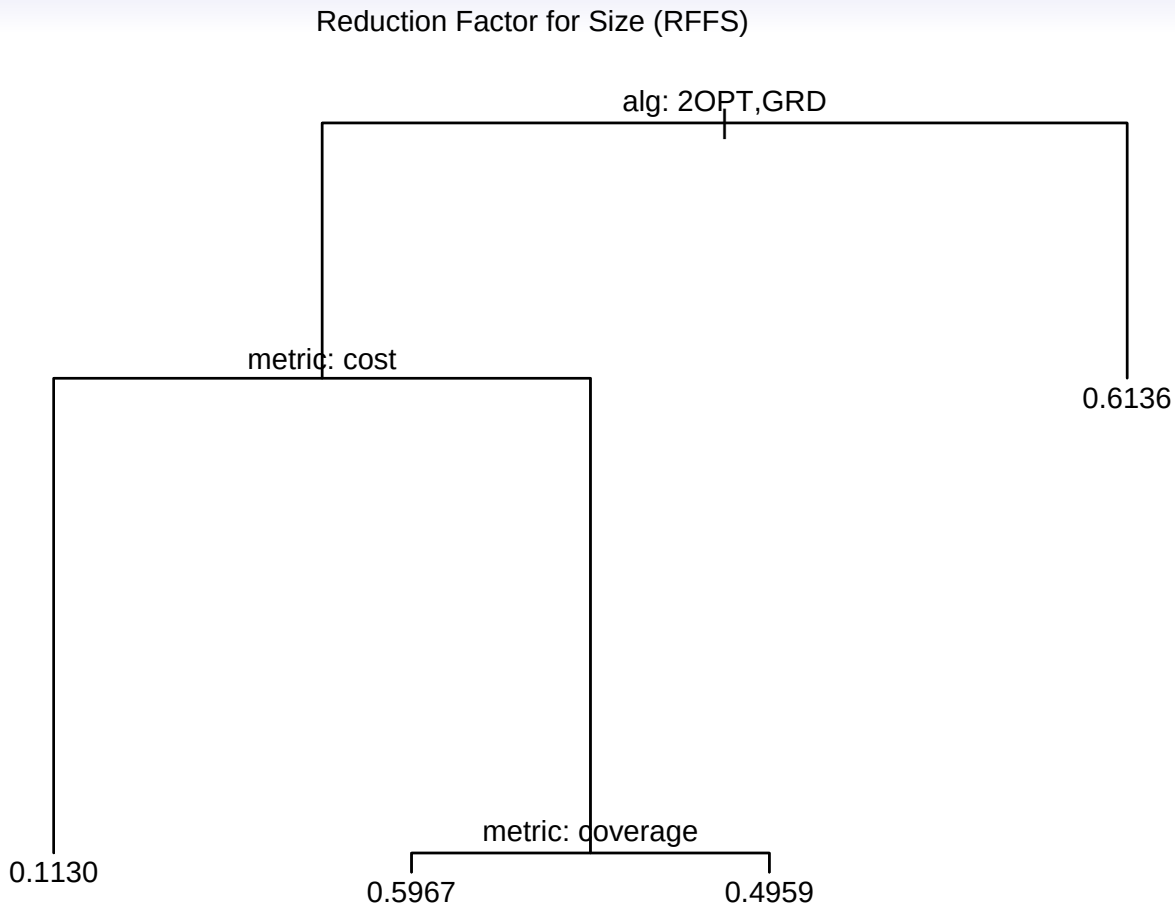
The myopic focus on **cost** leads to **low** RFFT values for 2OPT and GRD

Overview of RFFT Trends



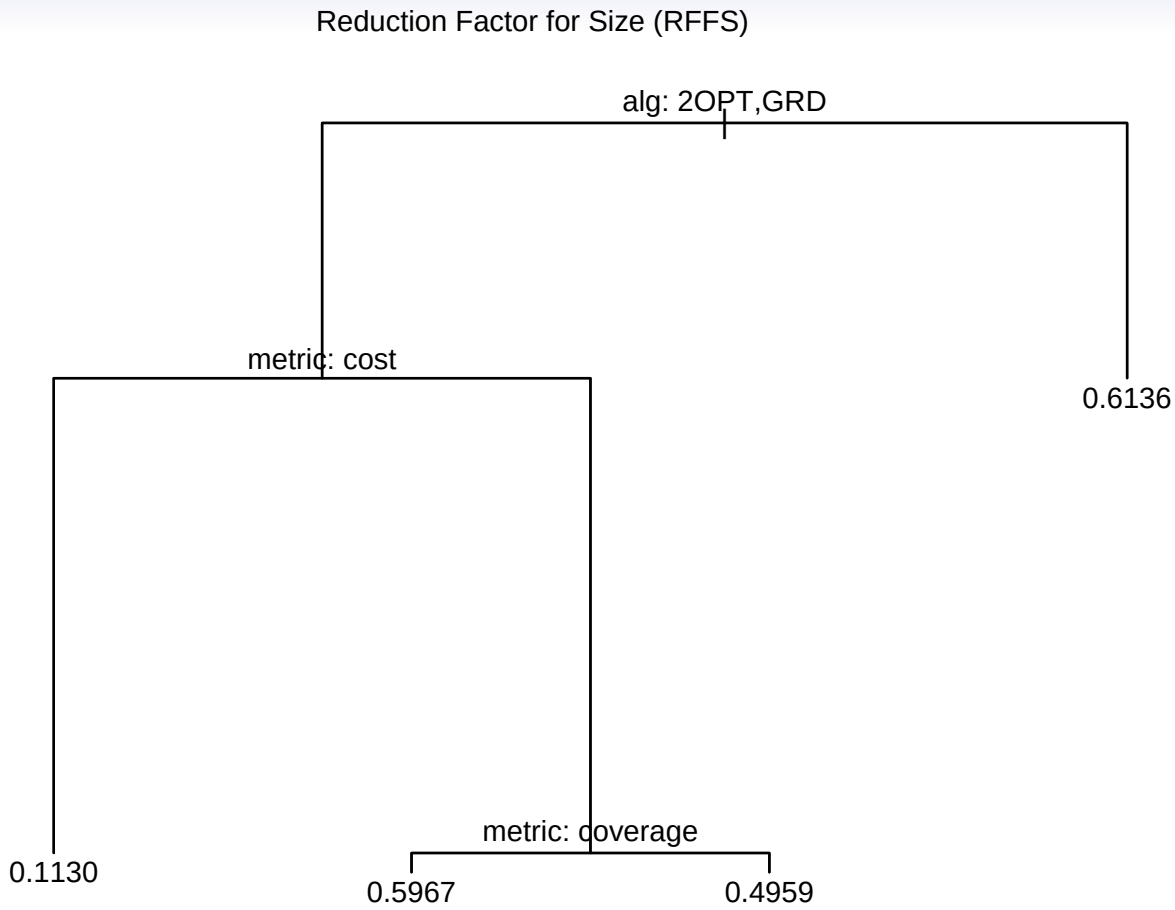
The myopic focus on **cost** leads to **low** RFFT values for 2OPT and GRD

Overview of RFFS Trends



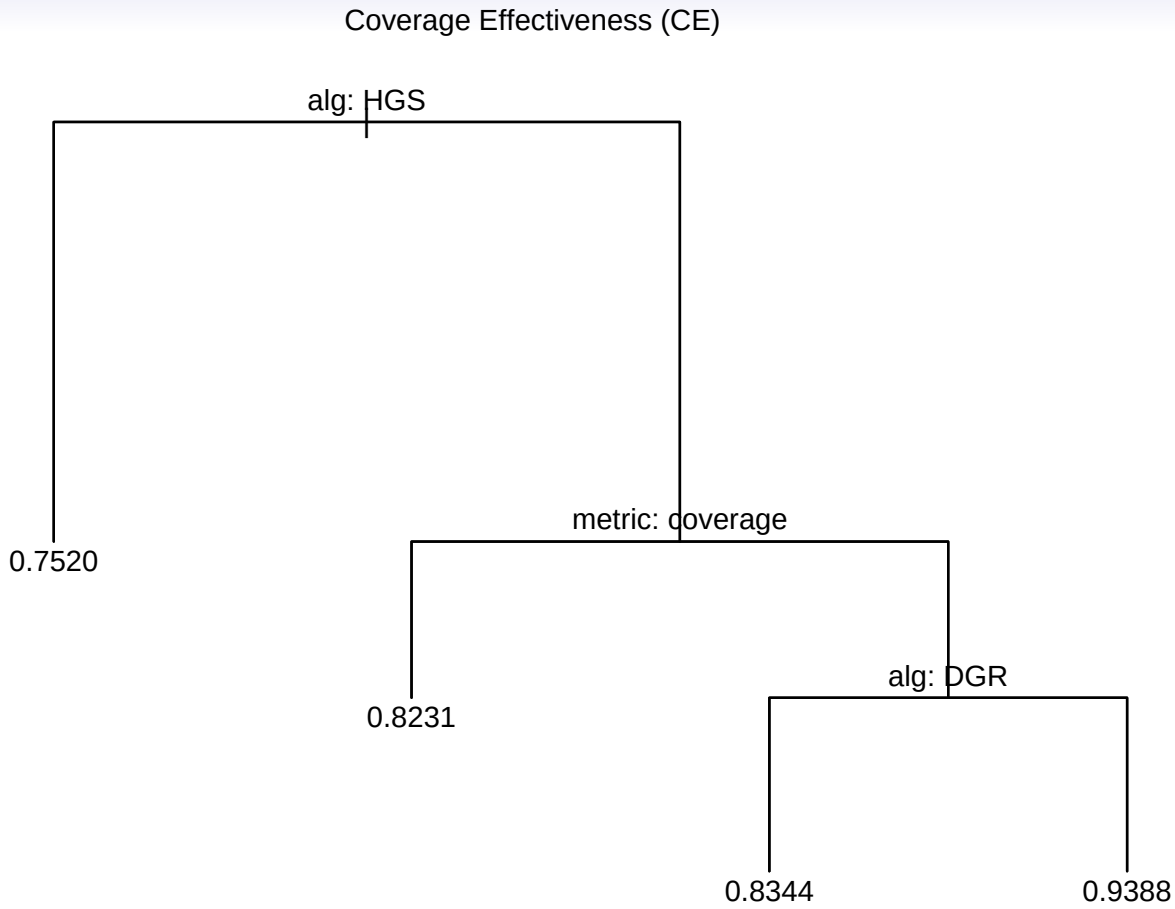
DGR and HGS are the **best** at creating test suites that **improve** RFFS

Overview of RFFS Trends



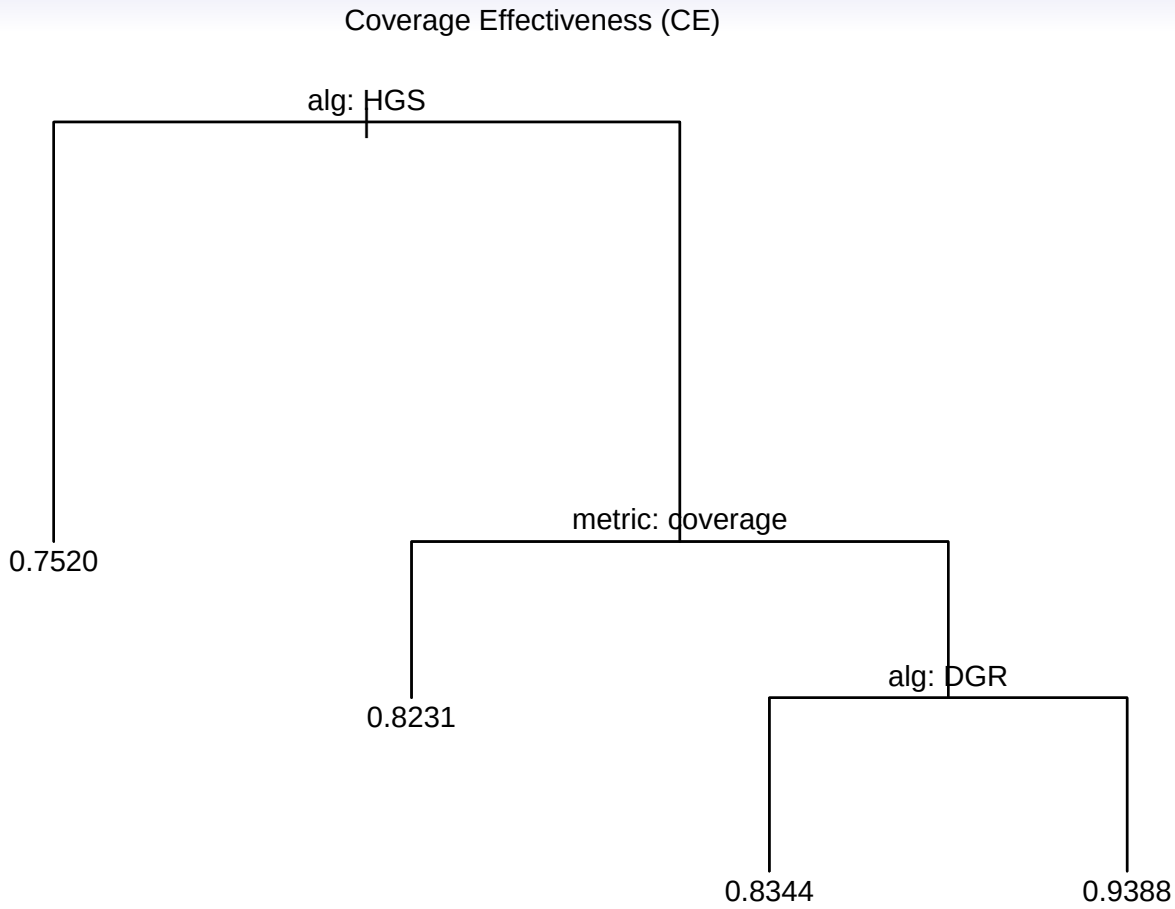
DGR and HGS are the **best** at creating test suites that **improve** RFFS

Overview of CE Trends



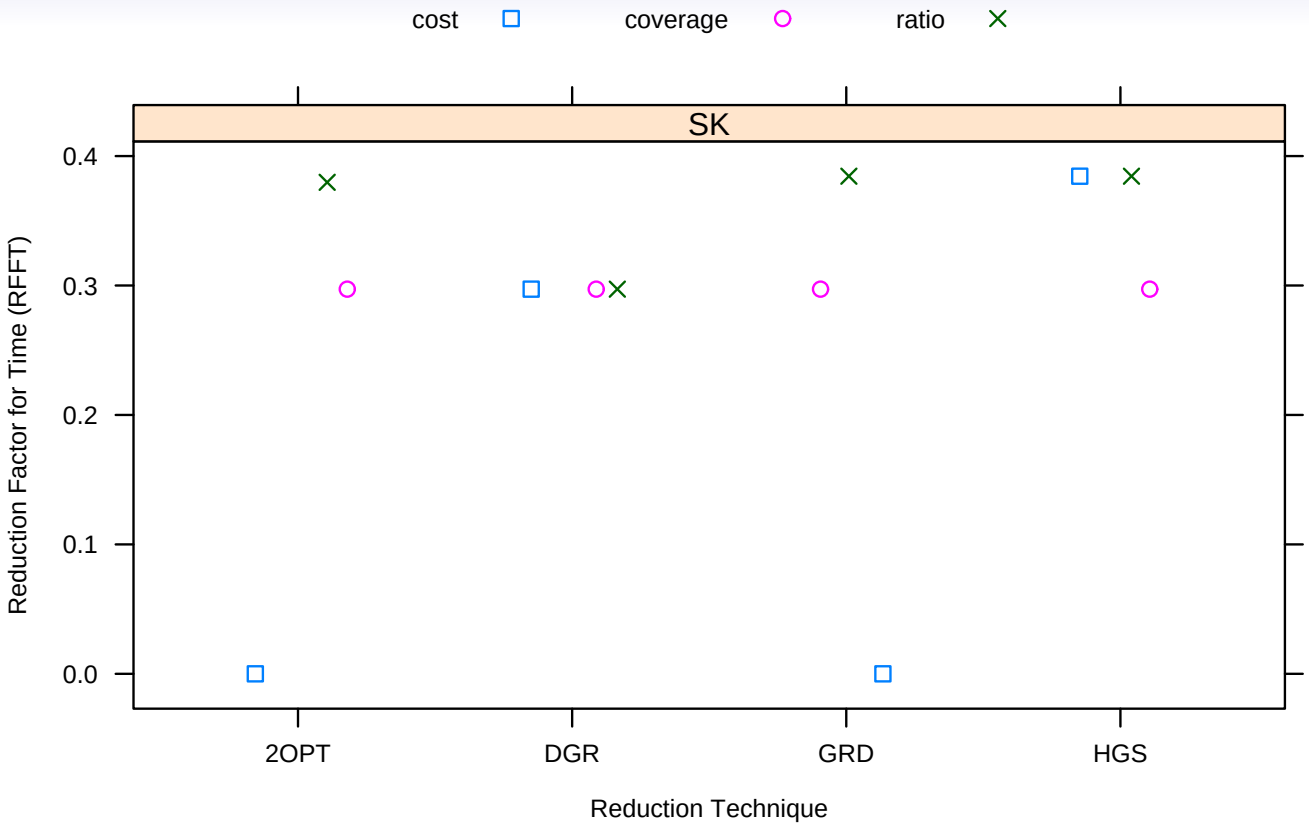
Using **ratio** and **cost** improves the CE of the prioritized test suite

Overview of CE Trends



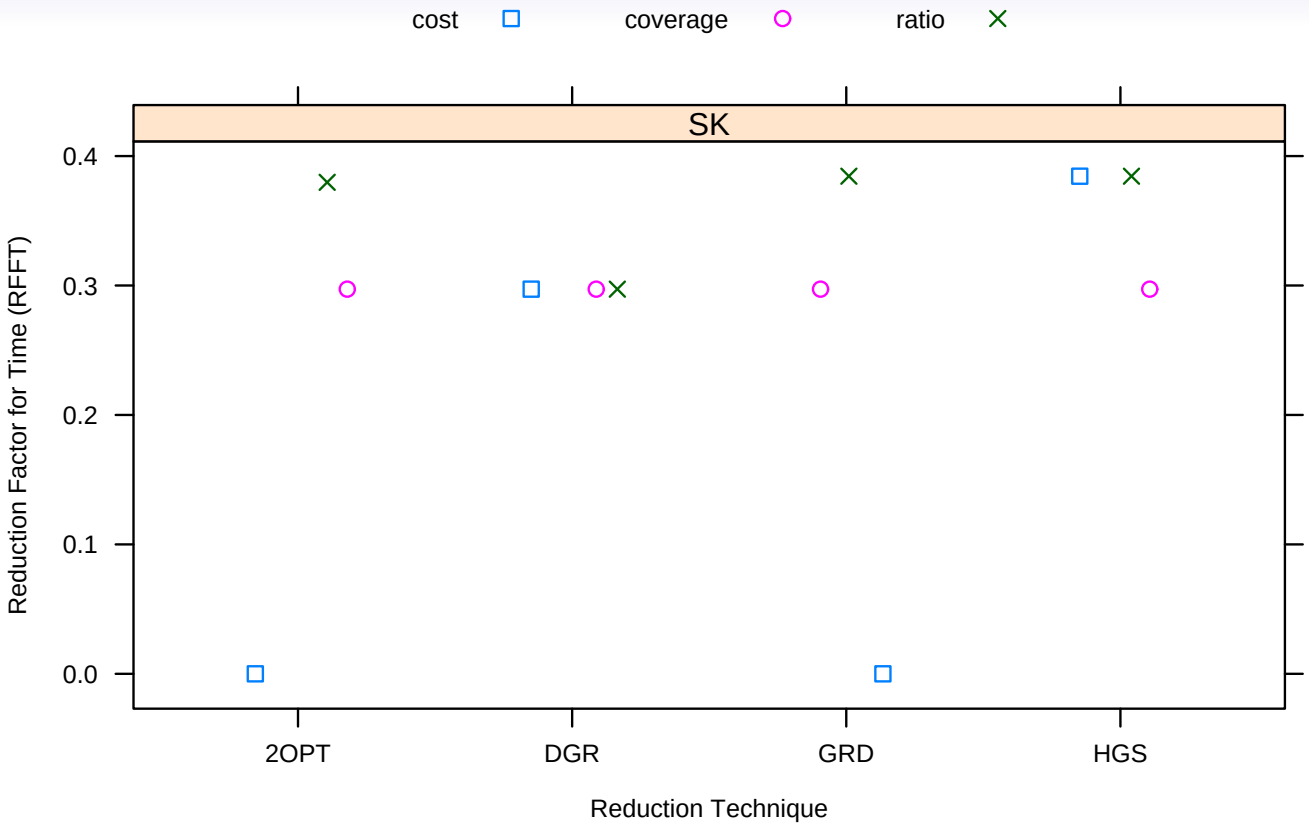
Using **ratio** and **cost** improves the CE of the prioritized test suite

Reduction Factor for Time - SK



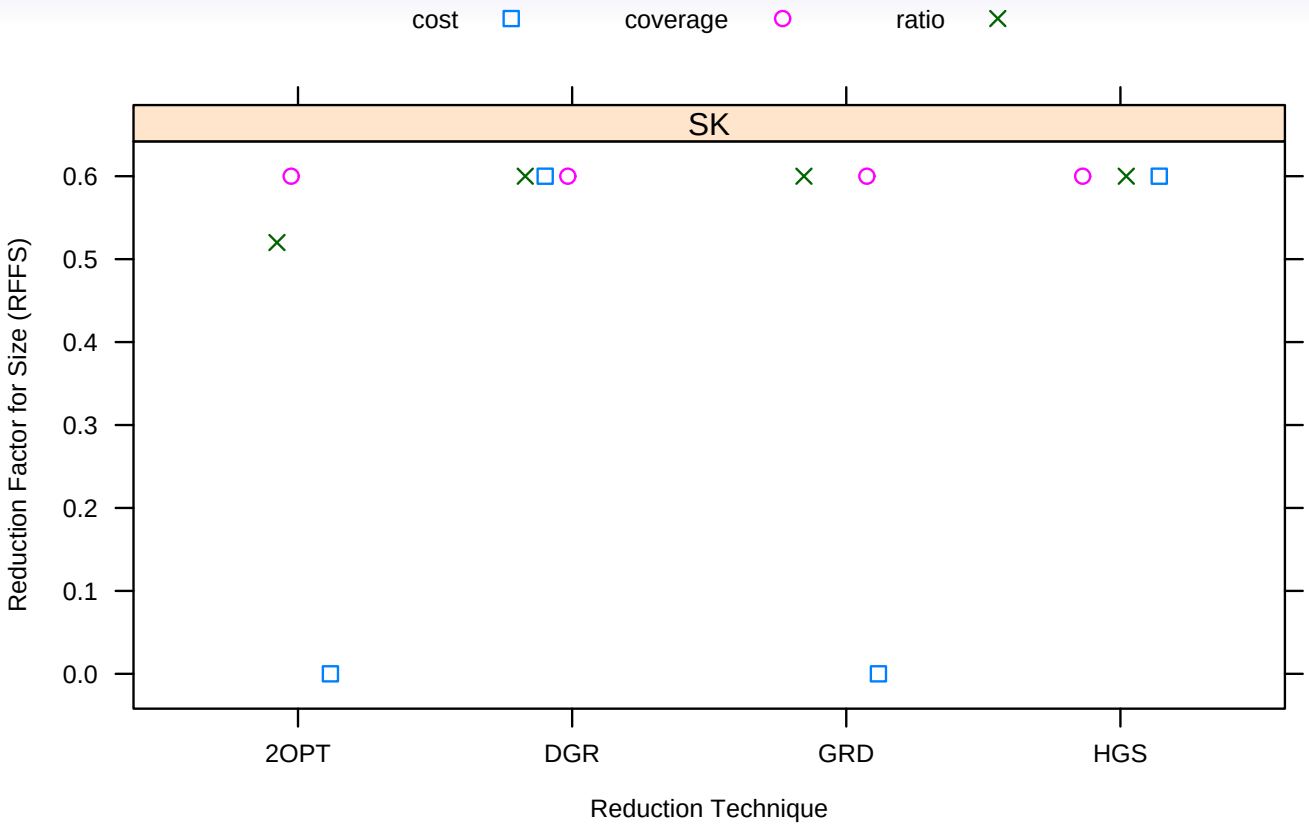
For 2OPT and GRD, **ratio** and **coverage** create the best test suites

Reduction Factor for Time - SK



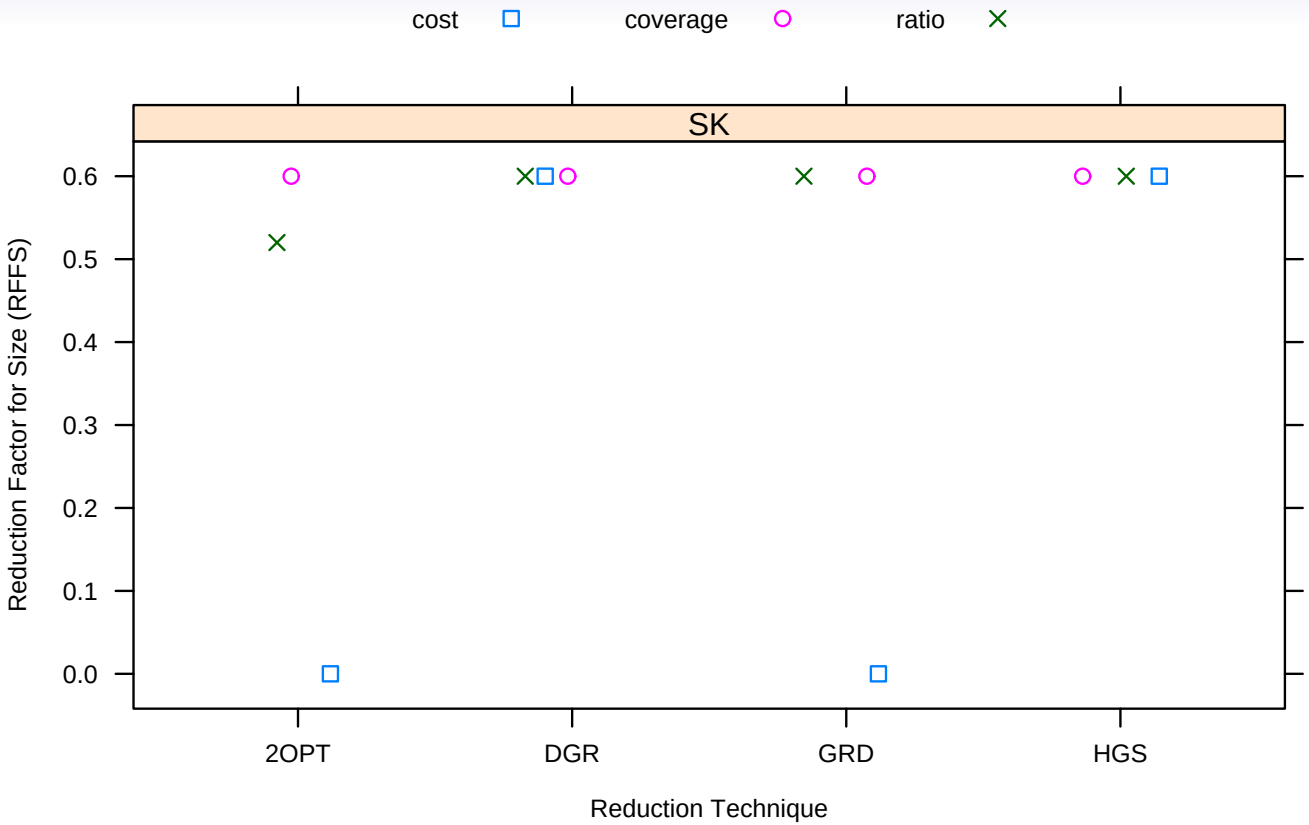
For 2OPT and GRD, **ratio** and **coverage** create the best test suites

Reduction Factor for Size - SK



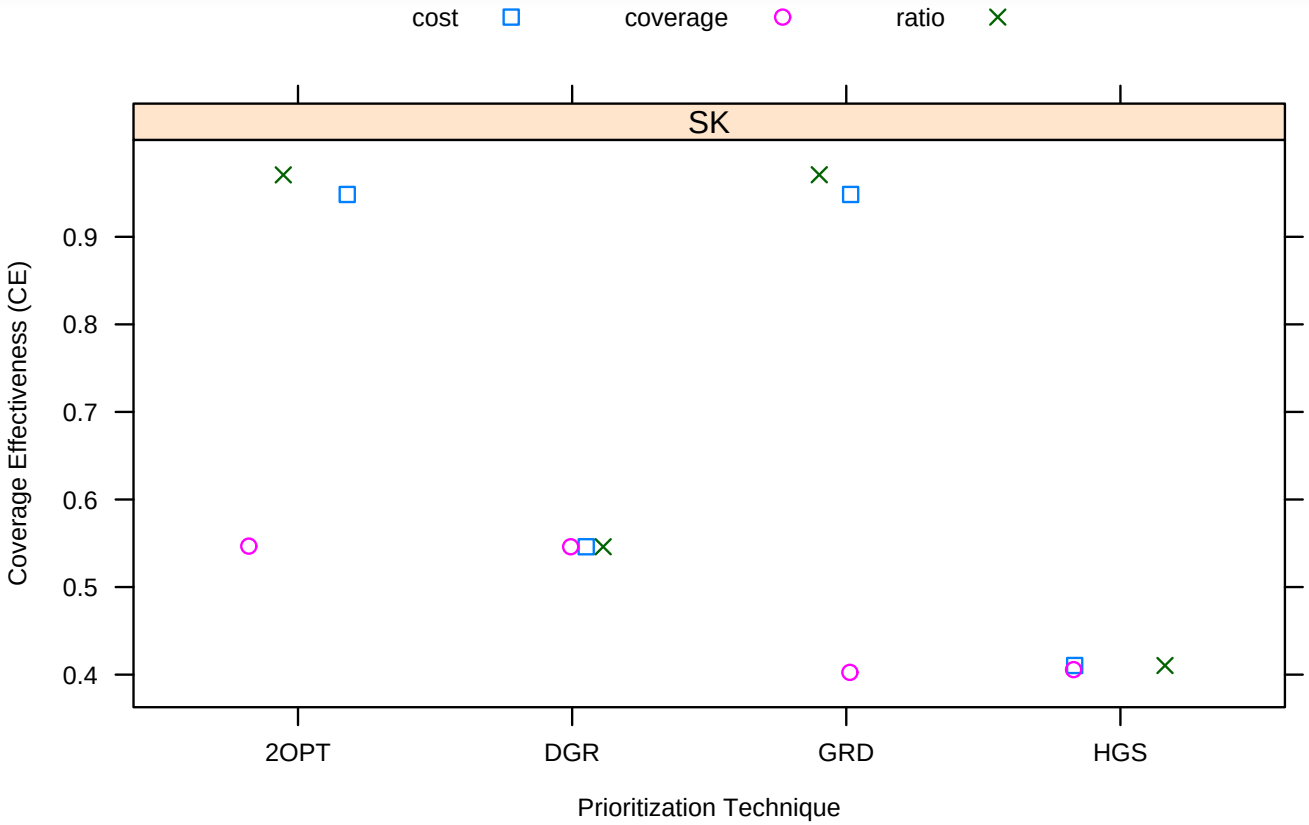
It is often **easy** to construct test suites with **high** RFFS values

Reduction Factor for Size - SK

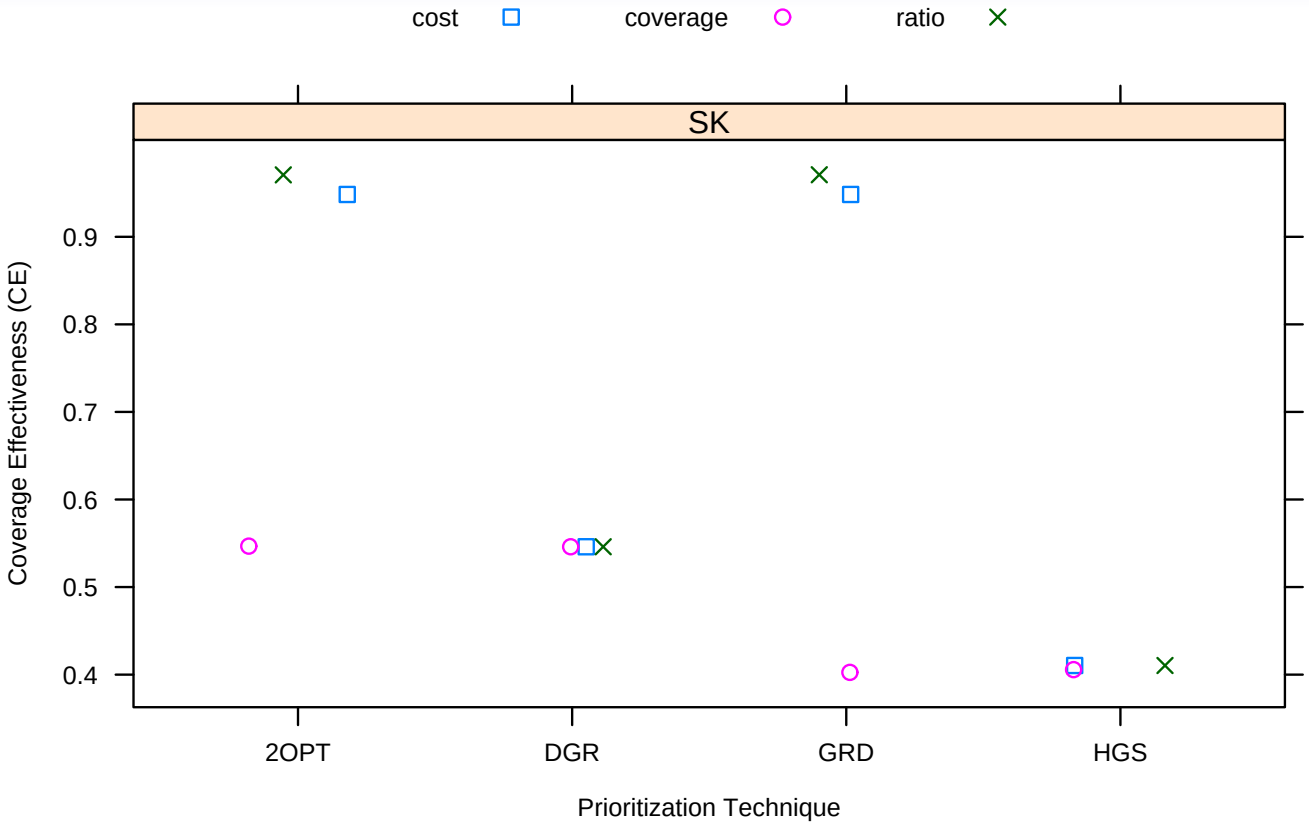


It is often **easy** to construct test suites with **high** RFFS values

Coverage Effectiveness Results - RP

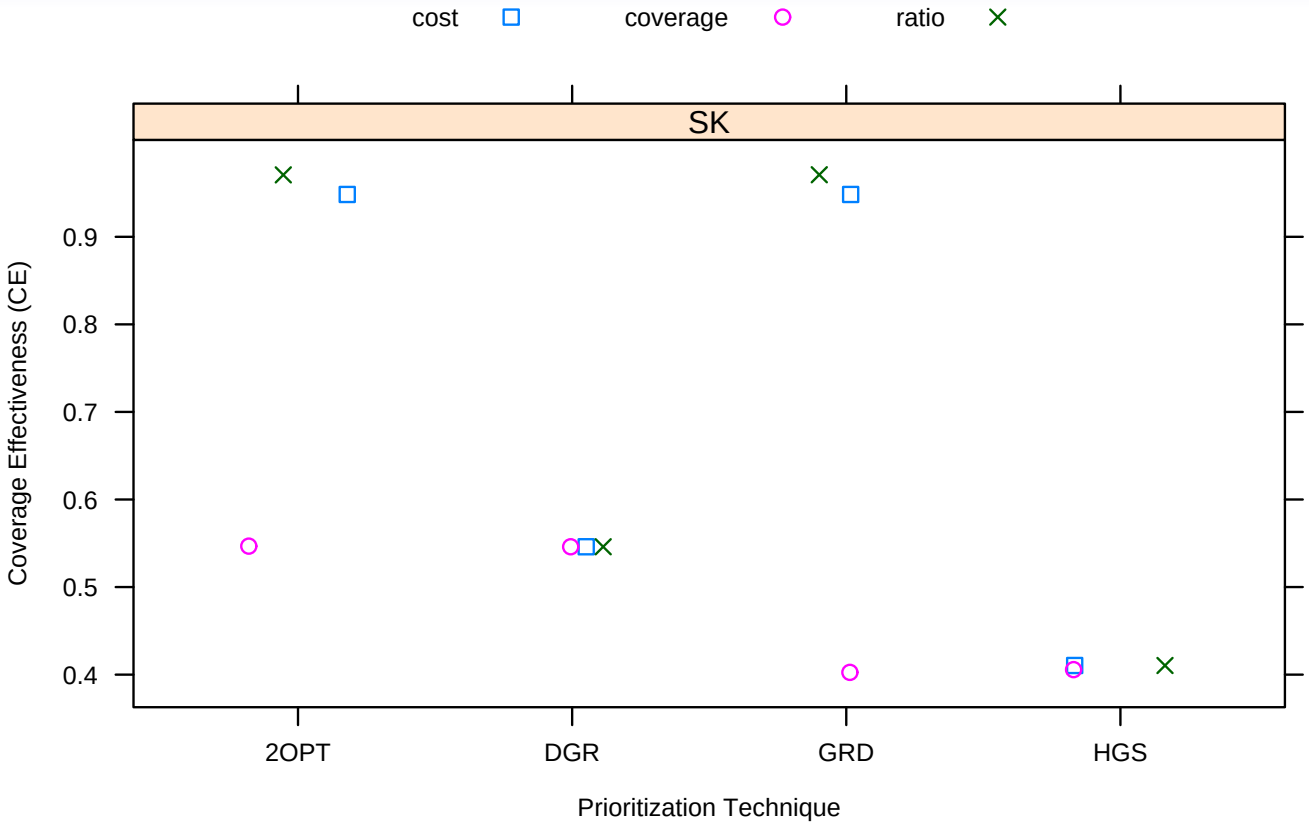


Coverage Effectiveness Results - RP



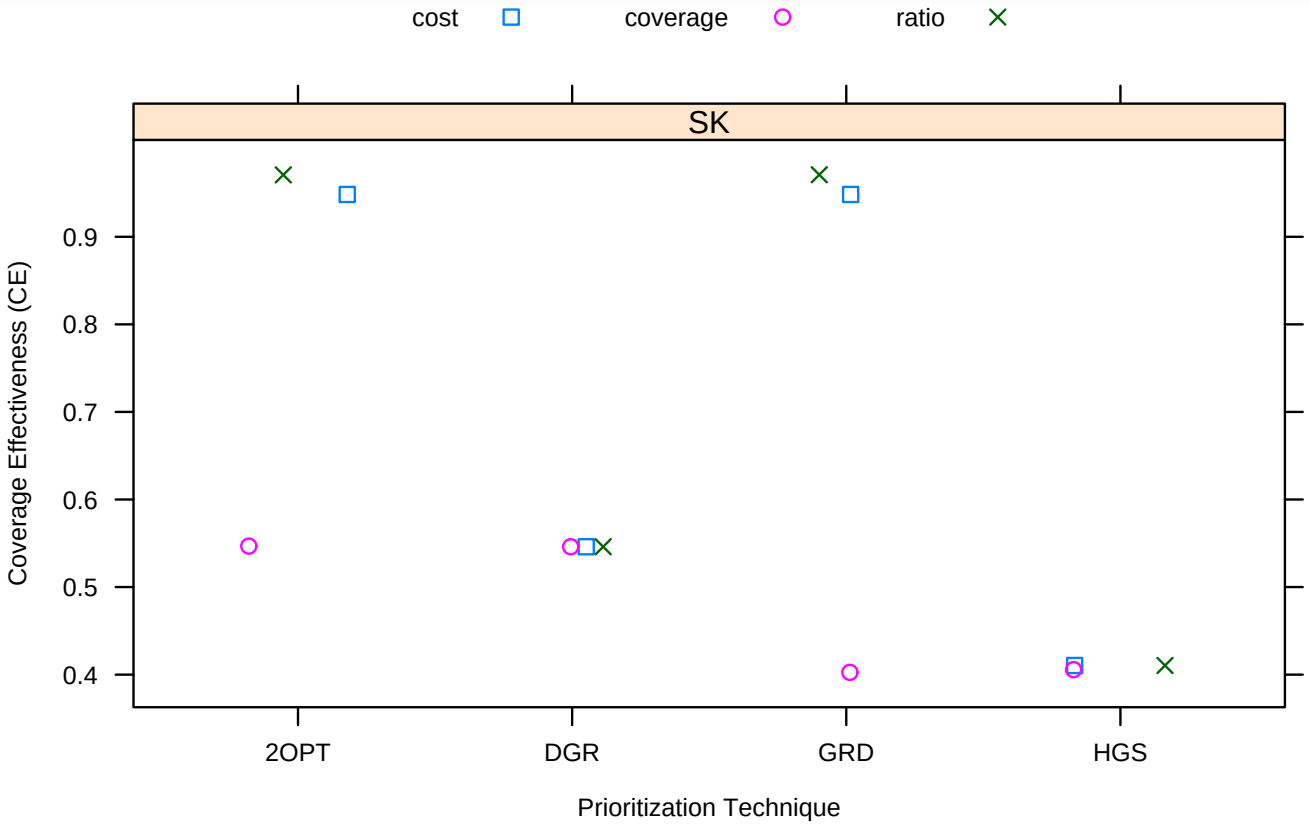
DGR and HGS exhibit **lackluster** performance when **reordering**

Coverage Effectiveness Results - RP



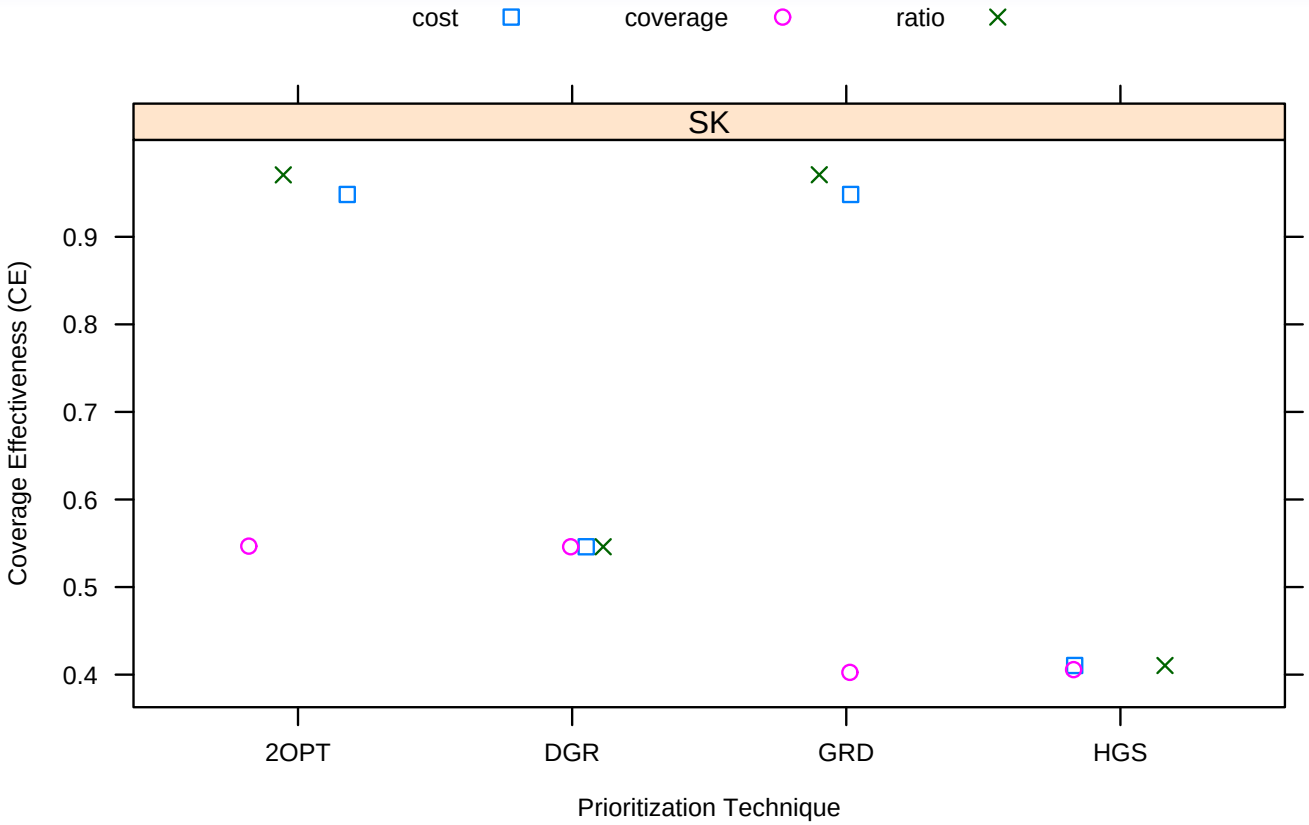
Greedily reordered test suites are **better** than randomly prioritized ones

Coverage Effectiveness Results - RP



Greedy fooling test suites cause GRD and DGR to make **low** CE suites

Coverage Effectiveness Results - RP



2OPT uses **lookahead** and can construct high CE test prioritizations

Common Rate Scores

Application	*CommonRate*(Υ)
Reminder	0.700
ReduceAndPrioritize	0.361
Sudoku	0.571
TransactionManager	0.450
DataStructures	0.171
GradeBook	0.747
JDepend	0.606
LoopFinder	0.500
Average	0.513

Common Rate Scores

Application	*CommonRate*(ϒ)
Reminder	0.700
ReduceAndPrioritize	0.361
Sudoku	0.571
TransactionManager	0.450
DataStructures	0.171
GradeBook	0.747
JDepend	0.606
LoopFinder	0.500
Average	0.513

Common Rate Scores

Application	*CommonRate*(Υ)
Reminder	0.700
ReduceAndPrioritize	0.361
Sudoku	0.571
TransactionManager	0.450
DataStructures	0.171
GradeBook	0.747
JDepend	0.606
LoopFinder	0.500
Average	0.513

Common Rate Scores

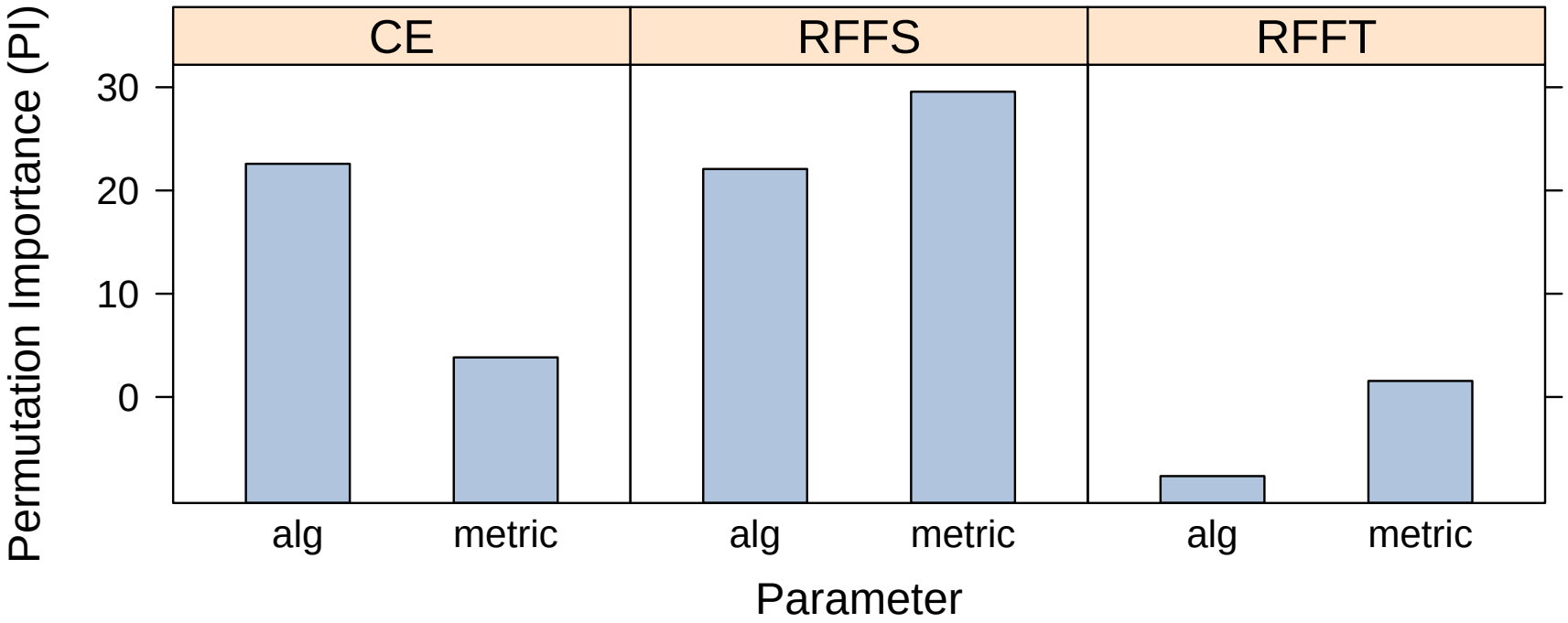
Application	*CommonRate*(Υ)
Reminder	0.700
ReduceAndPrioritize	0.361
Sudoku	0.571
TransactionManager	0.450
DataStructures	0.171
GradeBook	0.747
JDepend	0.606
LoopFinder	0.500
Average	0.513

Common Rate Scores

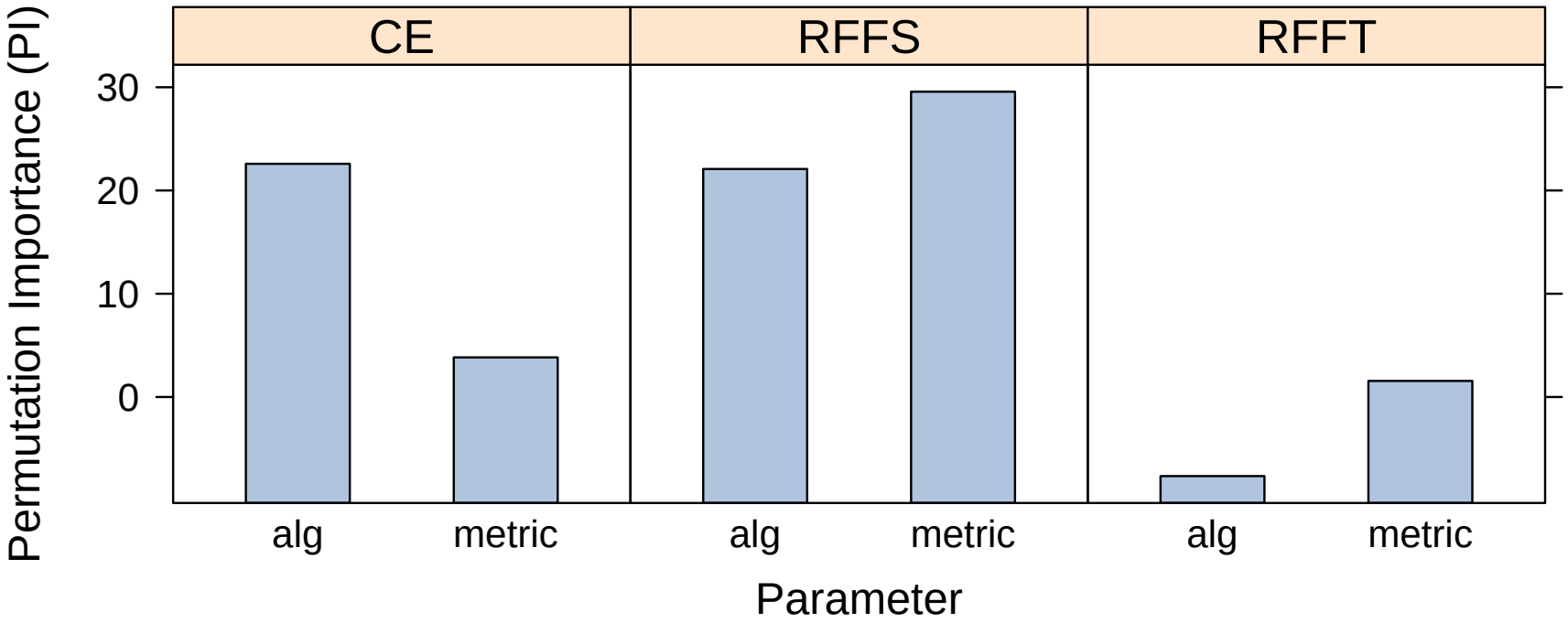
Application	*CommonRate*(Υ)
Reminder	0.700
ReduceAndPrioritize	0.361
Sudoku	0.571
TransactionManager	0.450
DataStructures	0.171
GradeBook	0.747
JDepend	0.606
LoopFinder	0.500
Average	0.513

Value of the common rate is relatively **stable** across methods

Parameter Importance Values

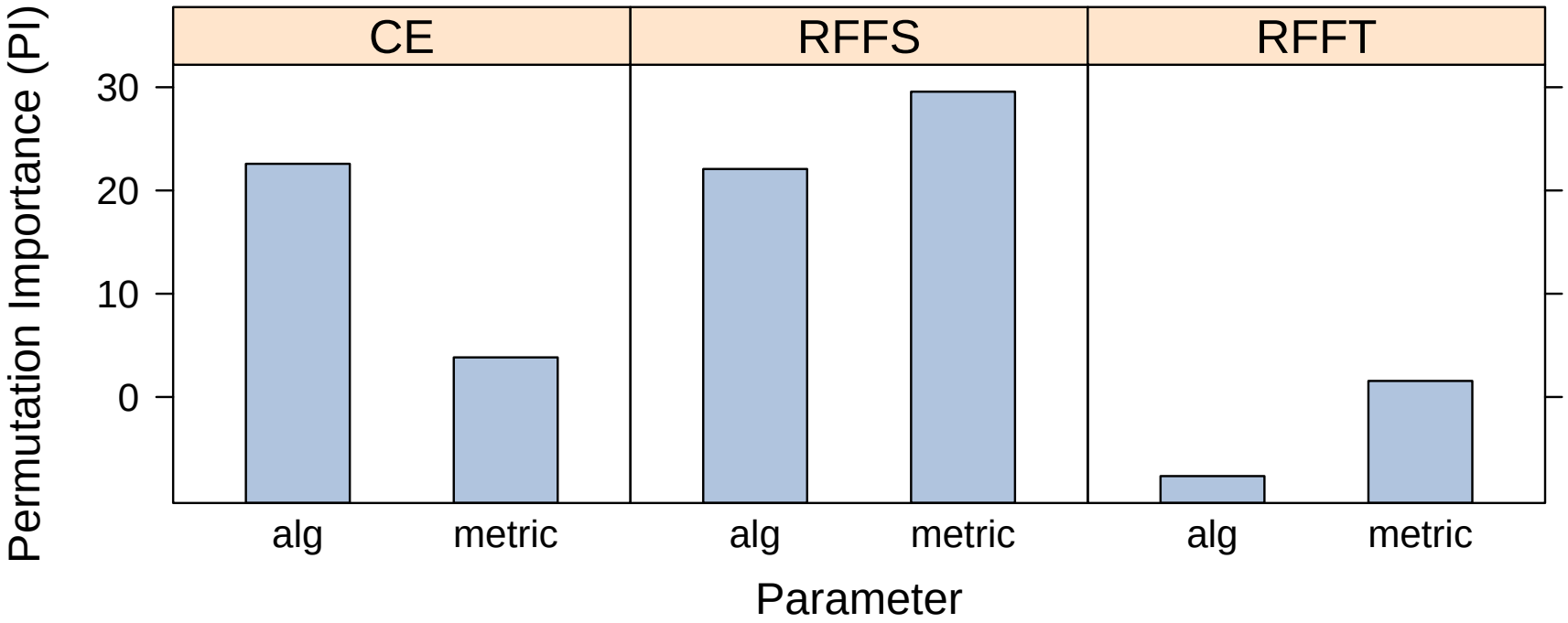


Parameter Importance Values



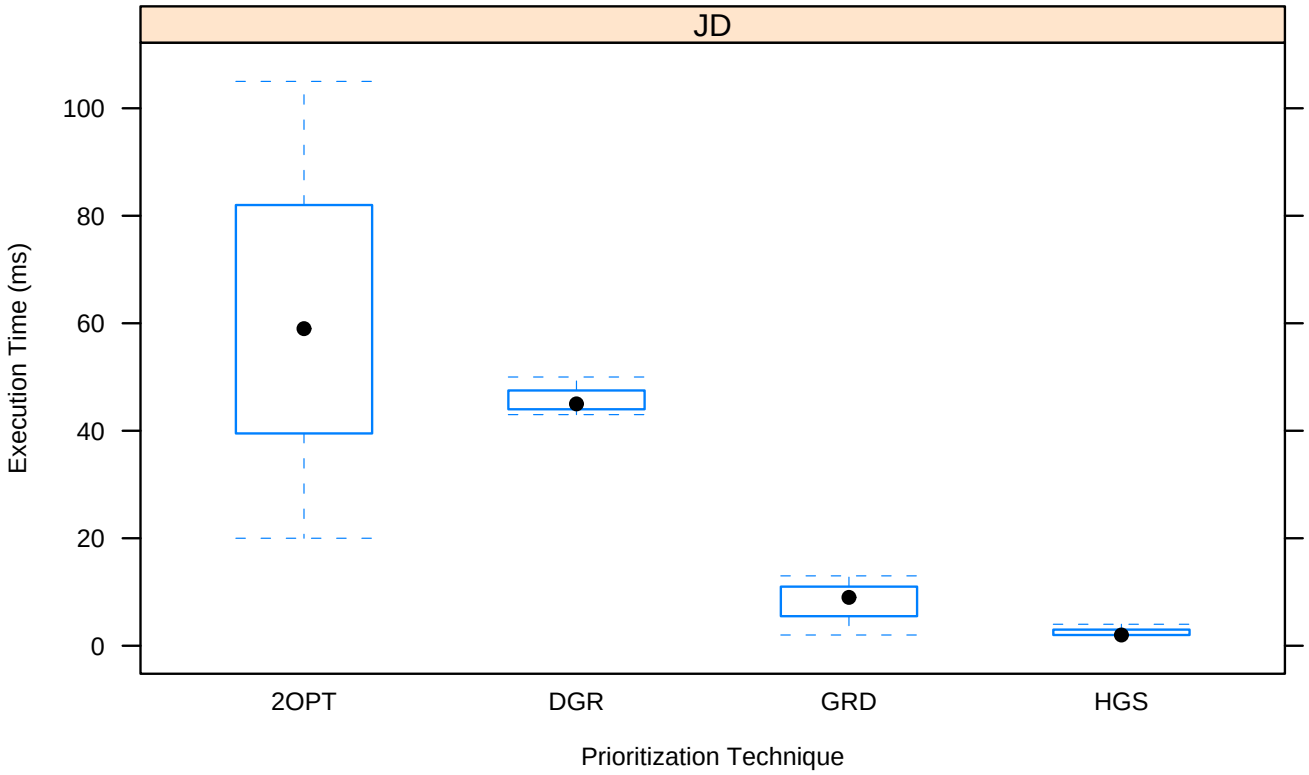
Algorithm choice is most important for improving the CE of ordering

Parameter Importance Values



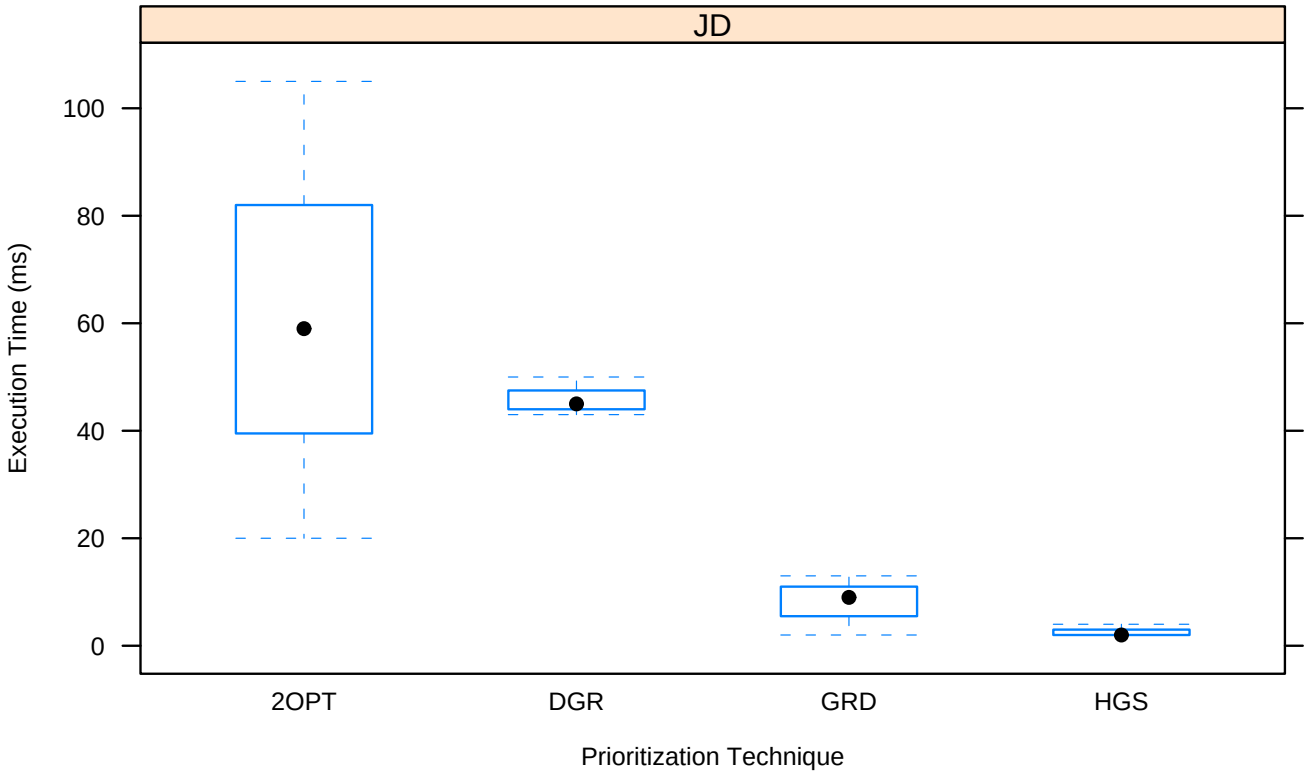
Greedy choice **metric** has the greatest impact on the test suite reducers

Efficiency Measurements



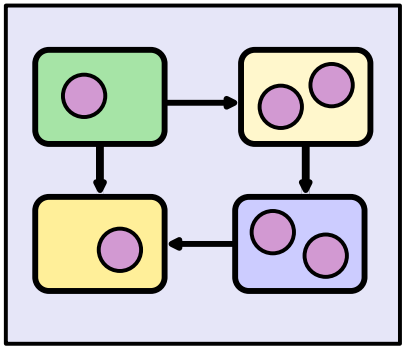
For the **chosen** case study applications, the techniques are **fast**

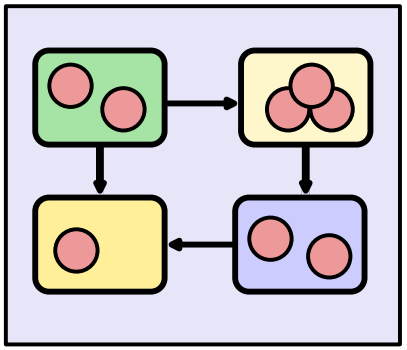
Efficiency Measurements



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

Alternative Evaluation Metrics Like APFD



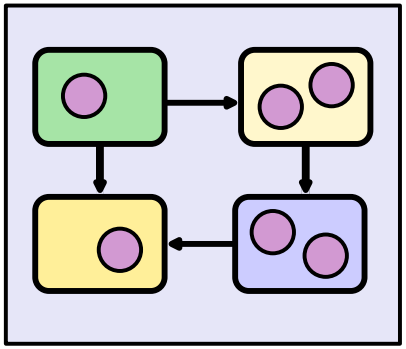


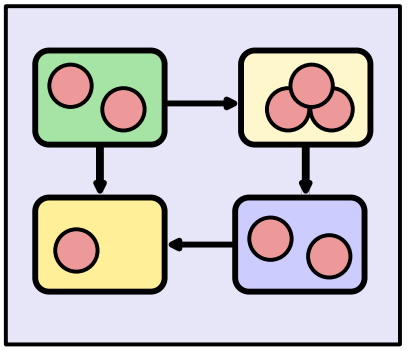
Mutation Faults

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.

Alternative Evaluation Metrics Like APFD



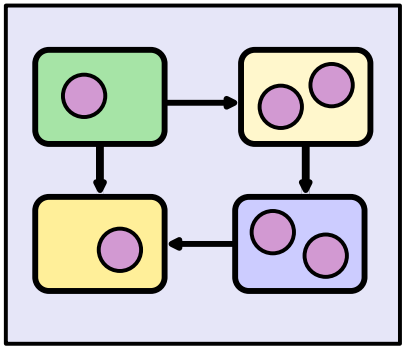


Mutation Faults

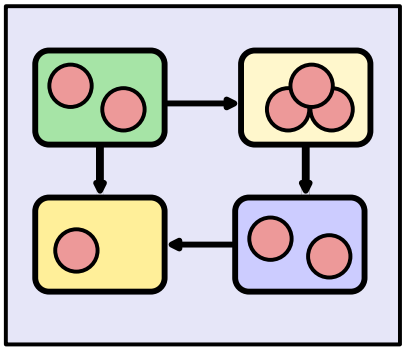
Real Faults

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.

Alternative Evaluation Metrics Like APFD



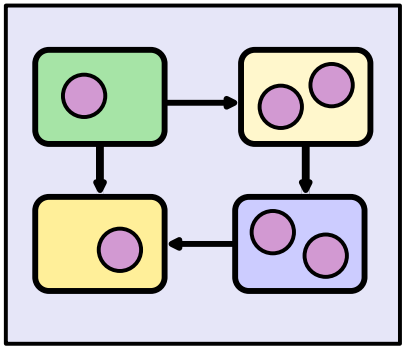
Mutation Faults

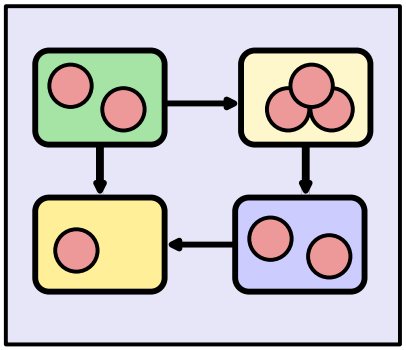


Real Faults

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.

Alternative Evaluation Metrics Like APFD



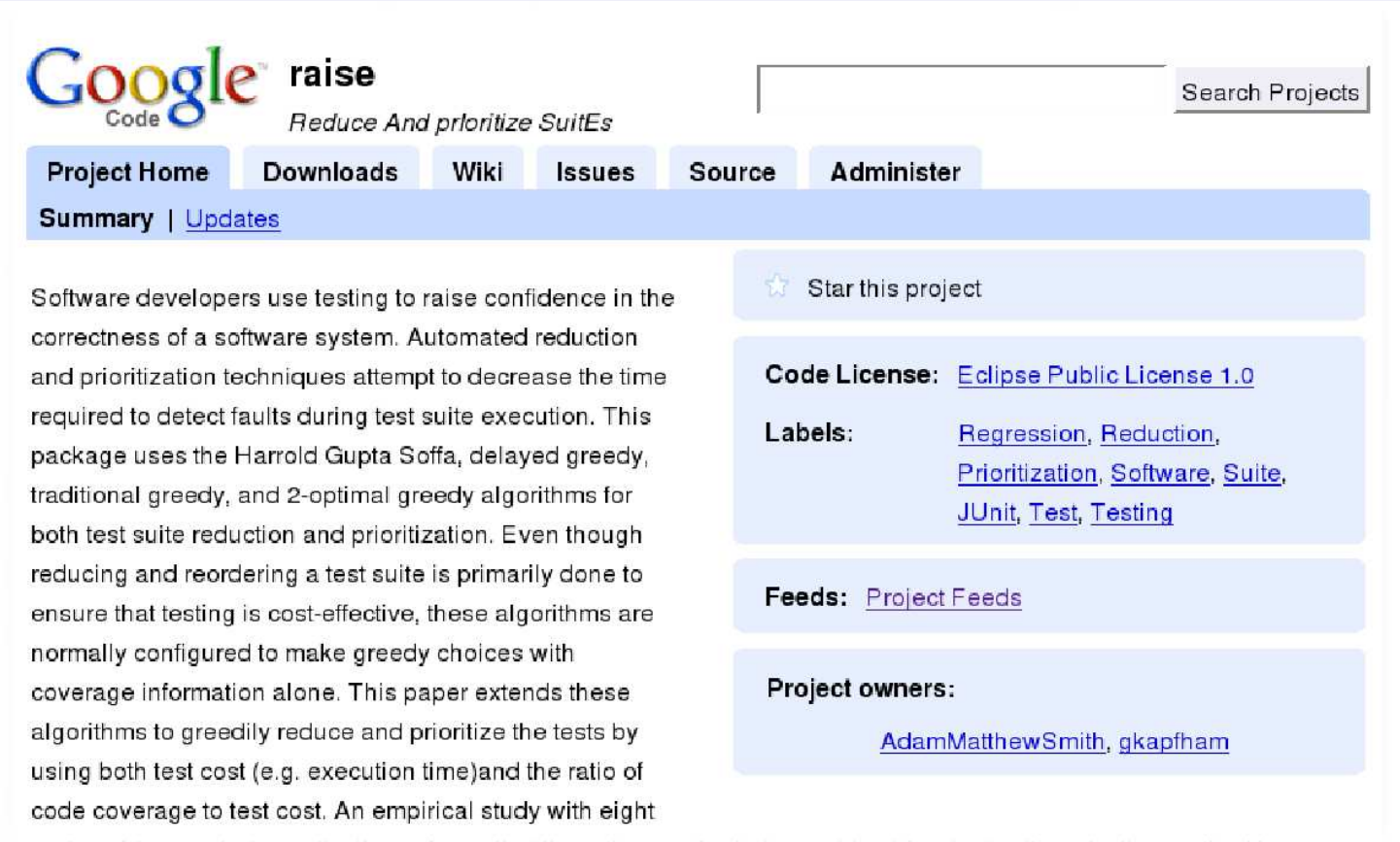


Mutation Faults

Real Faults

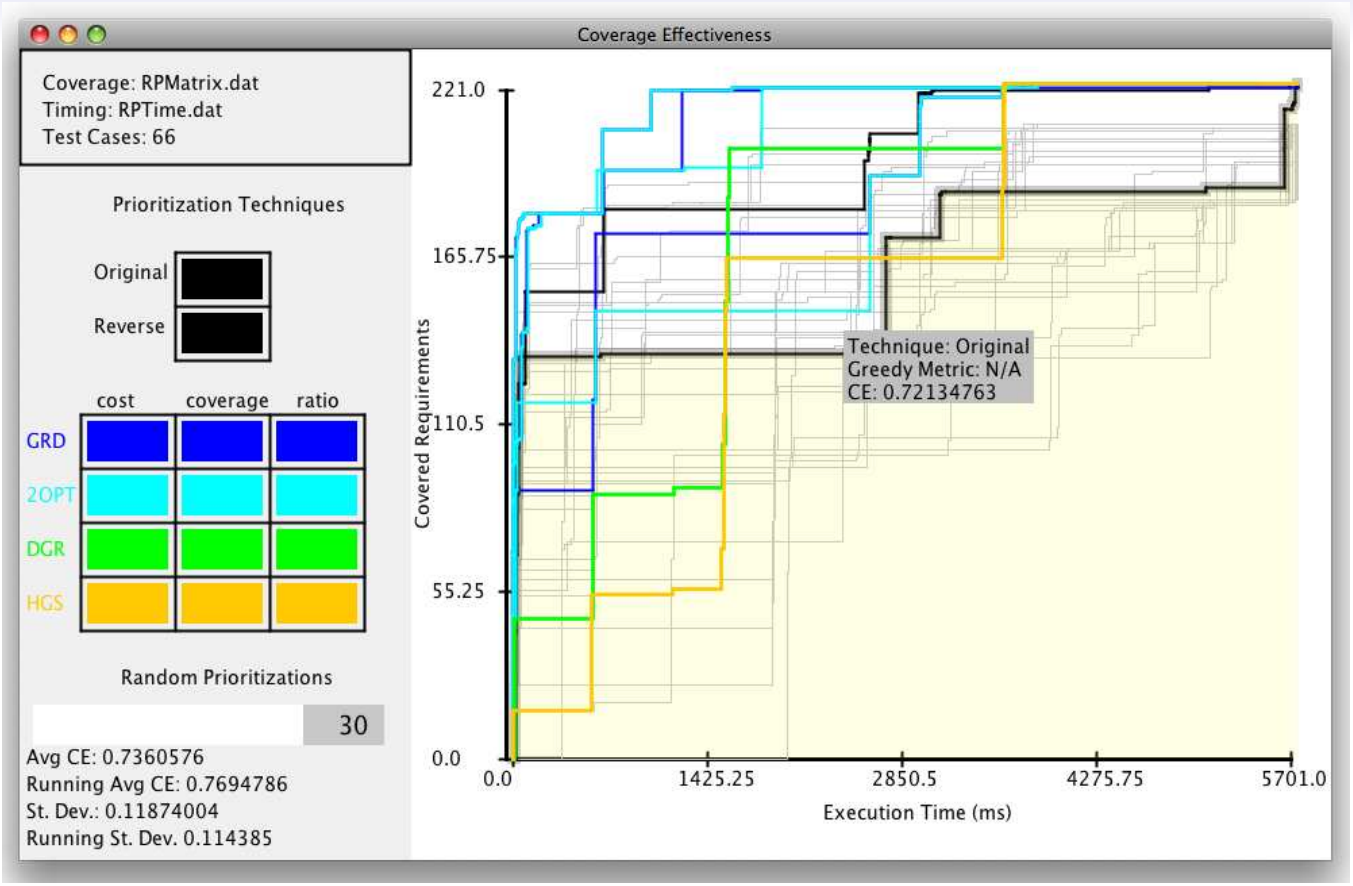
Use **mutation** and **real** faults to support the calculation of fault detection effectiveness (**FDE**) and average percentage of faults detected (**APFD**). Consider **search-based** testing methods.

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



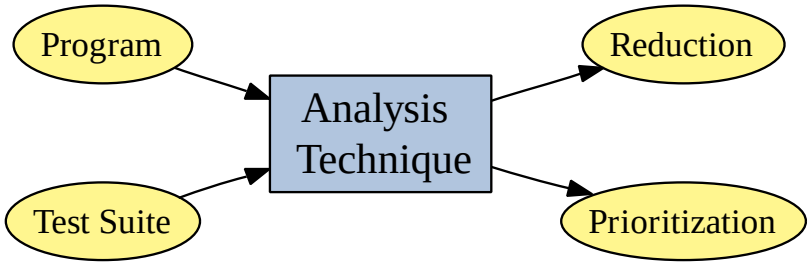
http://raise.googlecode.com/ provides tools, data sets, and resources

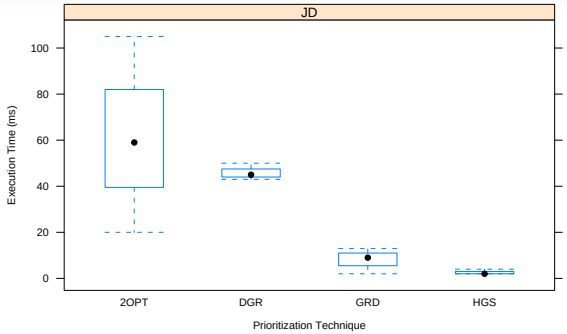
RAISE - *R*educe *A*nd pr*I*ortize Suit*E*s



Interactive visualization methods enable testers to find **best** ordering

Concluding Remarks





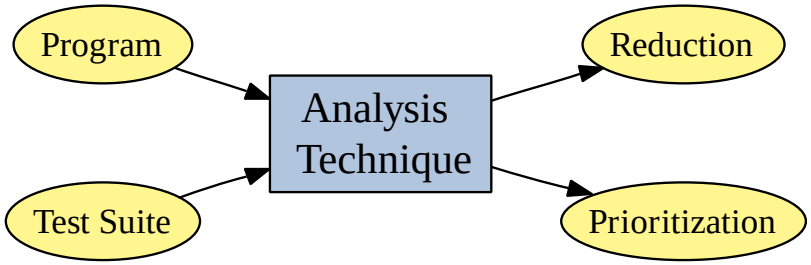
Regression Testing Techniques

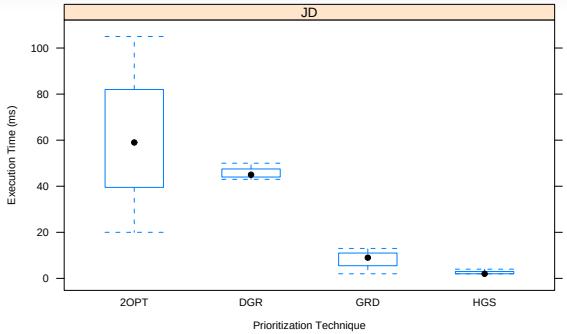
Detailed Empirical Results

- **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/

Concluding Remarks





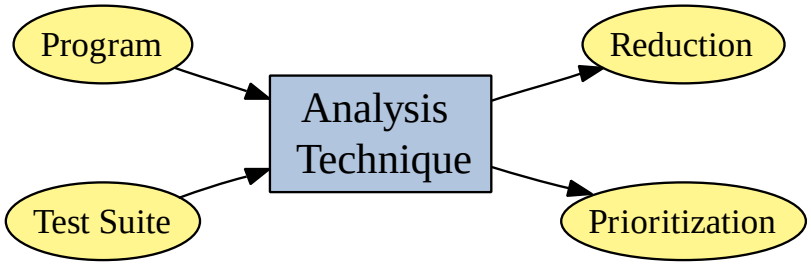
Regression Testing Techniques

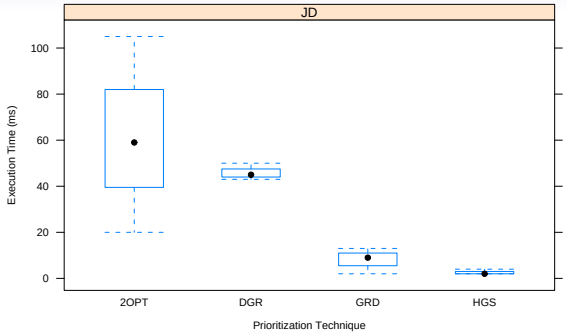
Detailed Empirical Results

- **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/

Concluding Remarks





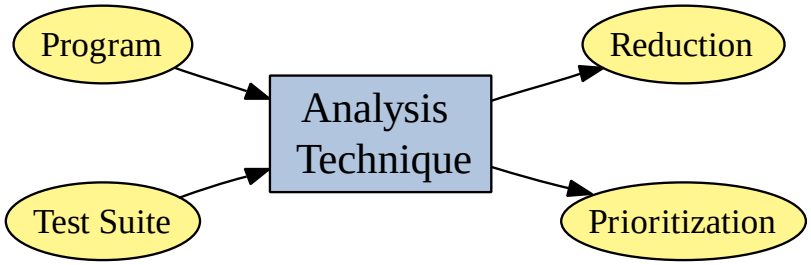
Regression Testing Techniques

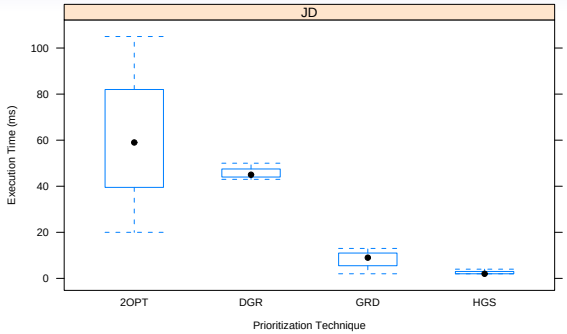
Detailed Empirical Results

- **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/

Concluding Remarks





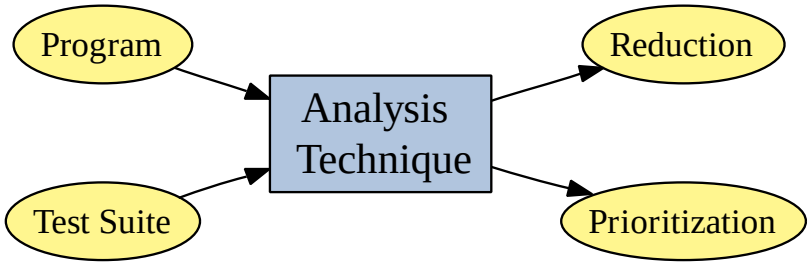
Regression Testing Techniques

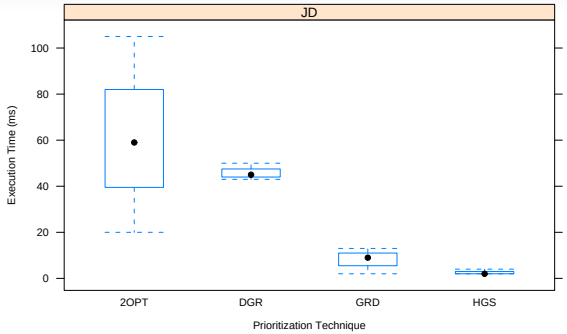
Detailed Empirical Results

- **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/

Concluding Remarks





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/