

# Reducing the Cost of Regression Testing by Identifying Irreplaceable Test Cases

---

**Chu-Ti Lin**, Dept. of Computer Sci. and Info. Eng., National Chiayi University, Taiwan

**Kai-Wei Tang**, Cloud System Software Institute, Institute for Information Industry, Taiwan

**Cheng-Ding Chen**, Dept. of Computer Sci. and Info. Eng., National Chiayi University, Taiwan

**Gregory M. Kapfhammer**, Dept. of Computer Science, Allegheny College, Meadville, PA

August 28, 2012

The Sixth International Conference on Genetic and Evolutionary Computing

# Outline

- Introduction
- Related work
- Reducing the execution cost of a test suite
- Experimental analysis
- Conclusion

# Introduction: Software Testing

- Software testing
  - To detect and isolate defects while implementing software systems.
- Test case
  - A set of input data and expected output results which are designed to exercise a specific software function or test requirement.

Test case \ Test requirement	Test requirement		
	$r_1$	$r_2$	$r_3$
$t_1$	●	●	
$t_2$			●
$t_3$		●	
$t_4$	●		

## Introduction: Test Suite

- It is difficult for a single test case to satisfy all of the specified test requirements.
- A **considerable number of test cases** are usually generated and collected in a test suite.

Test case \ Test requirement	$r_1$	...	$r_{3000}$
$t_1$	●		
$t_2$			●
:			
$t_?$	●		

# Introduction: Regression Testing

- In an attempt to ensure both the correctness of new code and its proper integration into the system, all test case in test suite  $T$  should be executed.

## Introduction: Test Suite Reduction

- To remove the redundant test cases while still ensuring that all test requirements are satisfied.

Test	$r_1$	$r_2$	$r_3$
$t_1$	●	●	
$t_2$			●
$t_3$		●	
$t_4$	●		

## Greedy Algorithm

- A commonly-used method for finding the **near-optimal** solution to the test suite reduction problem.
- It repeatedly removes the test  $t$  that has the maximum *Coverage*( $t$ ) from  $T$  to  $RS$  until all of the requirements are covered.
  - *Coverage*( $t$ ) is the number of uncovered test requirements satisfied by test case  $t$ .

## Greedy-based Algorithms

- Many test suite reduction algorithms are developed **based on Coverage metric**.
  - HGS algorithm proposed by Harrold et al. [4]
  - GE and GRE proposed by Chen and Lau [10]



## Reduction Using Greedy Algorithm

Test	Cost	$r_1$	$r_2$	$r_3$	Coverage( $t$ )
$t_1$	6	●	●		2
$t_2$	2			●	1
$t_3$	1		●		1
$t_4$	3	●			1

Greedy:  $RS = \{t_1, t_2\}$ , total cost = 8

Optimal solution:  $RS = \{t_2, t_3, t_4\}$ , total cost = 6

## Reduction with Ratio

- Ma et al. [11] and Smith and Kapfhammer [12] evaluated the test cases using

$$Ratio(t) = \frac{Coverage(t)}{Cost(t)}$$

where  $Cost(t)$  represents the execution cost of the test case  $t$ .

- It aims to reduce the cost of running a test suite.

## Reduction with Ratio

Test	Cost	$r_1$	$r_2$	$r_3$	$Ratio(t)$
$t_1$	6	●	●		0.67
$t_2$	2			●	0.5
$t_3$	1		●		1
$t_4$	3	●			0.33

## Reduction with Ratio

Test	Cost	$r_1$	$r_2$	$r_3$	$Ratio(t)$
$t_1$	6	●	-		0.17
$t_2$	2		-	●	0.5
$t_3$	1		-		-
$t_4$	3	●	-		0.33

## Reduction with Ratio

Test	Cost	$r_1$	$r_2$	$r_3$	$Ratio(t)$
$t_1$	6	●	-	-	0.17
$t_2$	2		-	-	-
$t_3$	1		-	-	-
$t_4$	3	●	-	-	0.33

## Reduction with Ratio

Test	Cost	$r_1$	$r_2$	$r_3$	$Ratio(t)$
$t_1$	6	-	-	-	0
$t_2$	2	-	-	-	-
$t_3$	1	-	-	-	-
$t_4$	3	-	-	-	-

$\text{Greedy}_{\text{WithRatio}} : RS = \{t_2, t_3, t_4\}$ , total cost = 6

## ReduceWithRatio Problems

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	$Ratio(t)$
$t_1$	4	•	•	•				0.75
$t_2$	7		•	•	•	•		0.57
$t_3$	3	•					•	0.67
$t_4$	4			•			•	0.50

## Problem of ReduceWithRatio

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	$Ratio(t)$
$t_1$	4	-	-	-				-
$t_2$	7	-	-	-	•	•		0.29
$t_3$	3	-	-	-			•	0.33
$t_4$	4	-	-	-			•	0.25



## Problem of ReduceWithRatio

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	$Ratio(t)$
$t_1$	4	-	-	-			-	-
$t_2$	7	-	-	-	•	•	-	0.29
$t_3$	3	-	-	-			-	-
$t_4$	4	-	-	-			-	0

## Problem of ReduceWithRatio

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	$Ratio(t)$
$t_1$	4	-	-	-	-	-	-	-
$t_2$	7	-	-	-	-	-	-	-
$t_3$	3	-	-	-	-	-	-	-
$t_4$	4	-	-	-	-	-	-	0

$\text{Greedy}_{\text{WithRatio}} : RS = \{t_1, t_2, t_3\}$ , total cost = 14

## Problem of ReduceWithRatio

$t_1$  is  
replaceable

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	$Ratio(t)$
$t_1$	4	•	•	•				0.75
$t_2$	7		•	•	•	•		0.57
$t_3$	3	•					•	0.67
$t_4$	4			•			•	0.50

Greedy<sub>WithRatio</sub> :  $RS = \{t_1, t_2, t_3\}$ , total cost = 14

Optimal solution :  $RS = \{t_2, t_3\}$ , total cost = 10

# Reduction Using Irreplaceability

- Concept:
  - Evaluating a test case by identifying if it is replaceable.
  - We posit that  $t$  has a higher *replaceability* with respect to  $r$  in this case
    - That is,  $t$  has a lower *irreplaceability* with respect to  $r$ .

## Evaluating the Irreplaceability

- The irreplaceability of  $t$  with respect to the requirement  $R = \{r_1, r_2, r_3, \dots, r_m\}$  can be defined as

$$\text{Irreplaceability}(t) = \frac{\sum_{i=1}^m \text{Contribution}(t, r_i)}{\text{Cost}(t)}$$

where

$$\text{Contribution}(t, r_i) = \begin{cases} 0, & \text{if } t \text{ cannot satisfy } r_i \\ \frac{1}{\text{the number of test cases that satisfy } r_i}, & \text{if } t \text{ satisfies } r_i \end{cases}$$

## Reduction with Irreplaceability

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	<i>Irreplaceability(t)</i>
$t_1$	4	•	•	•				0.33
$t_2$	7		•	•	•	•		0.40
$t_3$	3	•					•	0.33
$t_4$	4			•			•	0.21

## Reduction with Irreplaceability

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	<i>Irreplaceability</i> ( $t$ )
$t_1$	4	•	-	-	-	-		0.13
$t_2$	7		-	-	-	-		-
$t_3$	3	•	-	-	-	-	•	0.33
$t_4$	4		-	-	-	-	•	0.13

## Reduction with Irreplaceability

Test	Cost	$r_1$	$r_2$	$r_3$	$r_4$	$r_5$	$r_6$	<i>Irreplaceability</i> ( $t$ )
$t_1$	4	-	-	-	-	-	-	0
$t_2$	7	-	-	-	-	-	-	-
$t_3$	3	-	-	-	-	-	-	-
$t_4$	4	-	-	-	-	-	-	0

Greedy<sub>WithIrreplaceability</sub>:  $RS = \{ t_2, t_3 \}$ , total cost = 10

Optimal solution:  $RS = \{ t_2, t_3 \}$ , total cost = 10

Greedy<sub>WithRatio</sub>:  $RS = \{ t_1, t_2, t_3 \}$ , total cost = 14

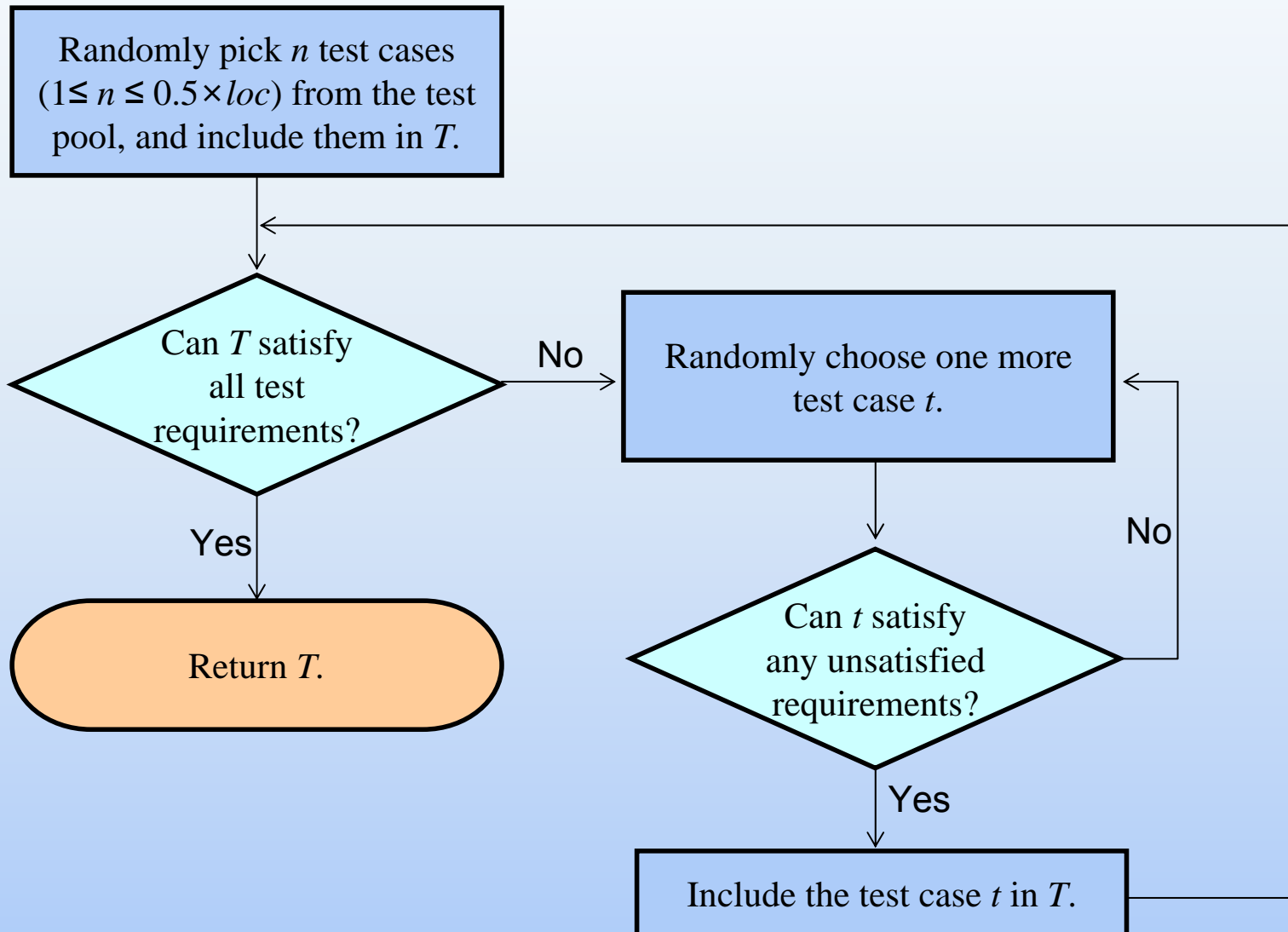


## Experimental Data Set

- The Siemens suite of programs from the SIR are frequently chosen benchmarks for evaluating test suite reduction methods [15].

Program	Test pool	Test requirements
printtokens	4,130	140
printtokens2	4,115	138
replace	5,542	126
schedule	2,650	46
schedule2	2,710	72
tcas	1,608	16
totinfo	1,052	44

# Experimental Setup



# Evaluating the Reduction Capability

- Criterion

$$SCR(T, RS) = \frac{Cost(T) - Cost(RS)}{Cost(T)} \times 100\%$$

where

$Cost(T)$ : the cost required to execute the original test suite  $T$ ;

$Cost(RS)$ : the cost associated with running the representative set  $RS$ .

## Experiment Result

Test Suite Program	Original	RS <sub>Greedy</sub>		RS <sub>WithRatio</sub>		RS <sub>WithIrreplaceability</sub>	
	Cost*	Cost*	SCR	Cost*	SCR	Cost*	SCR
Printtokens	914.67	117.32	87.17%	115.04	87.42%	81.73	91.06%
printtokens2	717.84	58.29	91.88%	56.19	92.17%	48.53	93.24%
Replace	1068.90	88.28	91.74%	81.06	92.42%	76.06	92.88%
Schedule	493.77	18.71	96.21%	16.35	96.69%	15.32	96.90%
schedule2	651.82	40.14	93.84%	28.60	95.61%	26.80	95.89%
Tcas	219.39	23.74	89.18%	21.53	90.19%	20.74	90.55%
Totinfo	690.97	52.15	92.45%	26.43	96.17%	26.14	96.22%

\* The cost is measured in millisecond(ms).

- Both ReduceWithIrreplaceability and ReduceWithRatio exhibit excellent cost reduction capabilities.
- The SCR scores of ReduceWithRatio are not as good as those of ReduceWithIrreplaceability.

# Summary of Contribution

- Key motivators
  - Most existing test suite reduction algorithms attempt to minimize the size of a regression test suite.
  - Reduction using Ratio metric does not always perform in a satisfactory manner.
- Method
  - Evaluating a test case by identifying if it is replaceable.
  - It repeatedly picks the test  $t$  that has the maximum *Irreplaceability* ( $t$ ).

# Summary of Contribution

- Empirical studies reveals that
  - Reduction using Irreplaceability is the best method for decreasing the cost of test suite execution.

# Future Work

