Group 16

DECREASING THE COST OF MUTATION TESTING WITH SECOND ORDER MUTANTS

83418 Afonso Feijão
96265 Luís Maçorano
111273 Leonor Côrte-Real

MUTATION TESTING

Mutation testing is a software testing technique used to evaluate the quality of test suites by introducing small changes to a program's source code.

Involves the following three stages:

- Mutant Generation
- Mutant Execution
- Result Analysis

The main goal is to write test cases that can detect these faults introduced by the mutants

EQUIVALENT MUTANTS

Mutants that, despite being syntactically different from the original program, behave identically.

Problem: Costly Process, detecting equivalent mutants is a costly task because it usually involves manual inspection to identify them among a large number of generated mutants.

Version	Code

Original	int sum(int a, int b) {
	return a + b;
	}

Mutant 1	int sum(int a, int b) {
	return a - b;
	}

Mutant 2	int sum(int a, int b) {
	return a * b;
	}

Mutant 3	int sum(int a, int b) {
	return a / b;
	}

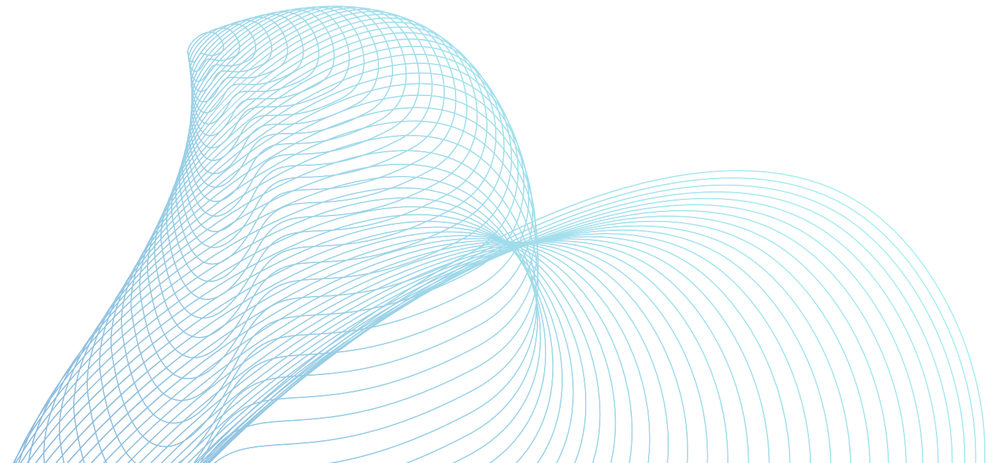
Mutant 4	int sum(int a, int b) {
	return a + b++;
	}

(a)

MUTATION SCORE

$$MS(P,T) = \frac{K}{(M - E)}$$, where:

P : program under test
T : test suite
K : number of killed mutants
M : number of generated mutants
E : number of equivalent mutants

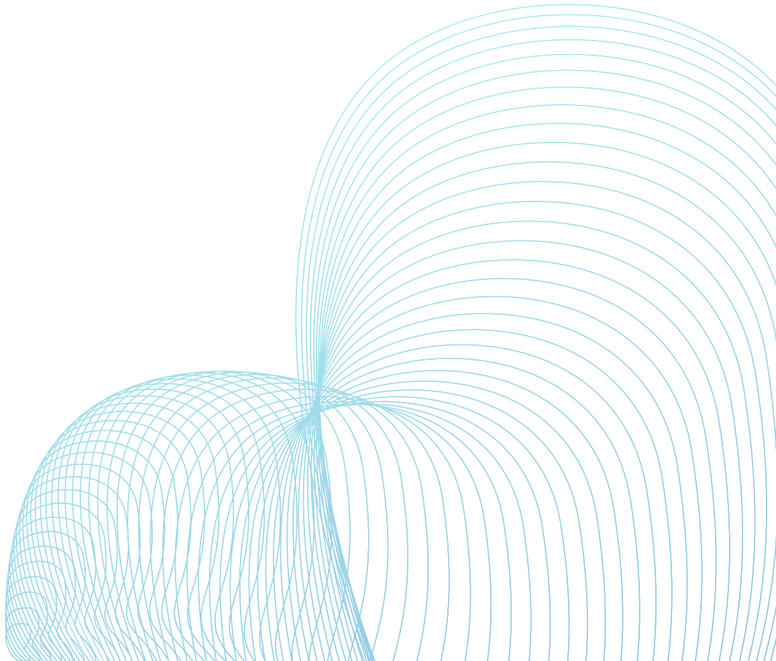


COST OF MUTATION TESTING

The three main stages of mutation testing — mutant generation, mutant execution, and result analysis — are **resource-intensive**:

- All test cases are executed against the original program and the mutants
- Equivalent mutants usually involves manual inspection

Proposed Technique to Reduce Costs: **Combining Mutants**



RELATED WORK FOR COST REDUCTION

Mutant Generation

Generation and Compilation of Mutants costs a lot!

Solution:

- Selective mutation
- Mutant Schemata Generation

Mutant Execution

Execution is exhausting

Solution:

- Non-standard Computer Architectures
- Weak Mutation

Result Analysis

Major drawback - manual detection is very costly.

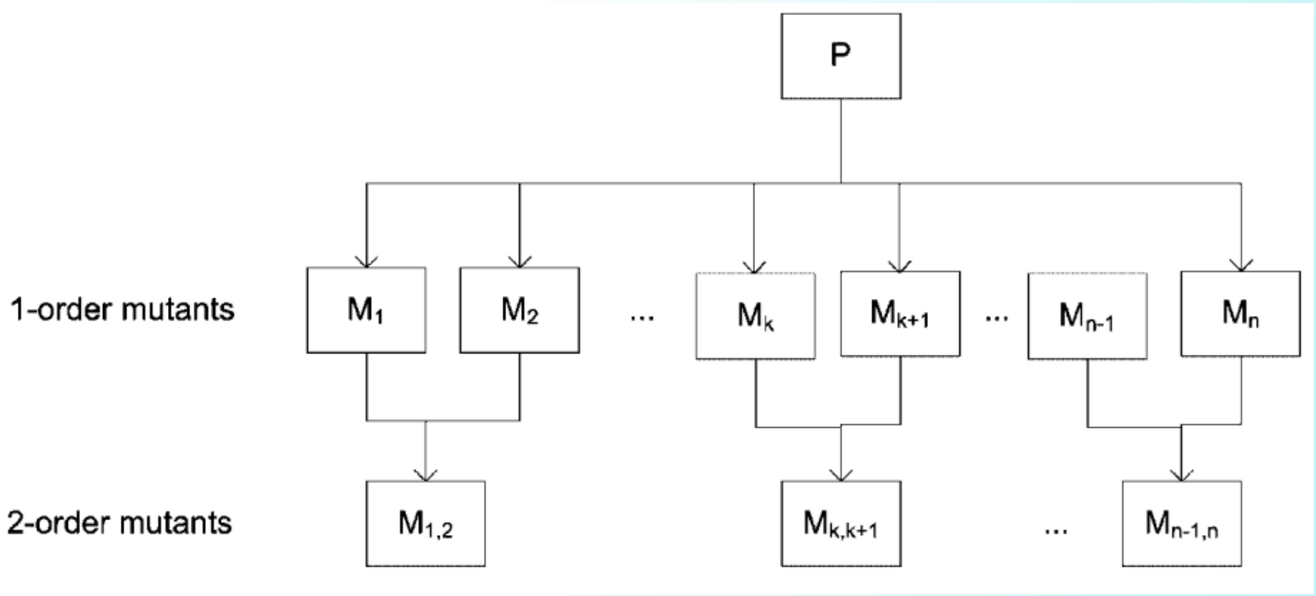
Solution:

- Annotate constraints on the program under test
- Compiler optimization techniques

DECREASING THE COST OF MUTATION

Idea: Combine First-Order Mutants into Second-Order Mutants

- **Reduces the number of mutants to nearly half of the original set of first-order mutants.**
- Reduces the number of Equivalent Mutants



DECREASING THE COST OF MUTATION

Idea: Combine First-Order Mutants into Second-Order Mutants

- Reduces the number of mutants to nearly half of the original set of first-order mutants.
- **Reduces the number of Equivalent Mutants**

M_i	M_j	$M_{i,j}$
Equivalent	Equivalent	Equivalent
Equivalent	Non-equivalent	Non-equivalent
Non-equivalent	Equivalent	Non-equivalent
Non-equivalent	Non-equivalent	Non-equivalent (very probably)

ALGORITHMS FOR SECOND-ORDER MUTATION

Last To First

First Order Mutants are in order in a file and the combination is done the first with the last, second with the previous to the last, etc.

Different Operators

This algorithm combines mutants proceeding from different mutation operators

Random Mix

Used as a baseline for comparison with the other two algorithms

Table III. MuJava mutants generated for the *Bisect* program.

AOIS_1	AOIS_35	AOIS_56	AOIU_11	AORB_1	ROR_1
AOIS_10	AOIS_37	AOIS_58	AOIU_12	AORB_10	ROR_10
AOIS_12	AOIS_38	AOIS_59	AOIU_13	AORB_3	ROR_4
AOIS_13	AOIS_39	AOIS_60	AOIU_2	AORB_5	ROR_6
AOIS_14	AOIS_4	AOIS_71	AOIU_3	AORB_7	
AOIS_15	AOIS_40	AOIS_72	AOIU_4		
AOIS_16	AOIS_41	AOIS_73	AOIU_6		
AOIS_17	AOIS_43	AOIS_74	AOIU_7		
AOIS_18	AOIS_44	AOIS_75	AOIU_8		
AOIS_19	AOIS_45	AOIS_76	AOIU_9		
AOIS_20	AOIS_47	AOIS_77			
AOIS_25	AOIS_48	AOIS_78			
AOIS_27	AOIS_5	AOIS_79			
AOIS_3	AOIS_52	AOIS_80			
AOIS_31	AOIS_54				

DESCRIPTION OF THE EXPERIMENTS

Two experiments were conducted to demonstrate the application of second-order mutants.

Experiment 1: Benchmark programs

- Applied the technique to a set of widely-studied programs in software testing research: Bisect, Bub, Find, Fourballs, Mid, and TriTyp.
- Each program is implemented as an isolated Java class, the first-order mutants were generated using MuJava and the test cases using the All Combinations strategy.

Experiment 2: Industrial software

- Demonstrated the technique's applicability to typical development scenarios, by applying it to three object classes from independent industrial systems: jtopas, jester and Tráfico.
- Selection criteria: availability of source code and test cases.

EXPERIMENT 1

The first table gives some information about the benchmark programs, all relative to First Order Mutation. The second table is about Second Order Mutation.

Comparing both tables it is seen that the number of second-order mutants is about 50% of first-order mutants.

The reduction in the mean percentage of equivalent mutants is greater (most notable in DifferentOperators due to its construction).

Table V. First-order mutants generated for the benchmark programs.

Program	LOC	No. of first-order mutants	Equivalent first-order mutants		No. of test cases
			Number	%	
Bisect	31	63	19	30.15	25
Bub	54	82	12	14.63	256
Find	79	179	0	0.00	135
Fourballs	47	212	44	20.75	96
Mid	59	181	43	23.75	125
TriTyp	61	309	70	22.65	216
			Mean	18.66	

Table VI. Second-order mutants after combining with the three algorithms.

Program	LastToFirst			DifferentOperators			RandomMix		
		Equivalent			Equivalent			Equivalent	
	No. of mutants*	No.	%	No. of mutants*	No.	%	No. of mutants*	No.	%
Bisect	32 (50.8)	5	15.63	44 (69.8)	5	11.36	32 (50.8)	2	6.25
Bub	41 (50)	0	0	44 (53.7)	0	0	40 (48.8)	1	2.5
Find	90 (50.3)	0	0	97 (54.2)	0	0	89 (49.7)	0	0
Fourballs	107 (50.4)	5	4.67	128 (60.4)	6	4.68	106 (50)	7	6.60
Mid	91 (50.3)	8	8.79	110 (60.8)	4	3.63	91 (50.3)	7	7.69
TriTyp	155 (50.2)	7	4.51	168 (54.4)	11	6.54	155 (50.2)	9	5.80
	Mean		5.6			4.4			4.8

KILLING MATRIX ANALYSIS

- For Bisect, Find, Fourballs and Mid, TC kill more than one mutant making it really difficult to write TC that discover only one of the faults of second order
- Trityp has 56 TC that don't kill any mutant.
- For Bub 174/255 = 0.68 (probability of discovering only one of the faults). Second-order mutants technique should be carefully applied.

Table VIII. Number of first-order mutants killed (KM) by test cases (TC) in the benchmark programs.

Bisect		Bub		Find		Mid				TriTyp			
KM	TC	KM	TC	KM	TC	KM	TC	KM	TC	KM	TC	KM	TC
17	3	1	174	2	1	11	1	40	3	0	56	86	2
18	5	2	1	3	75	14	4	41	4	3	48	88	1
20	1	25	1	4	20	17	2	42	1	4	48	89	1
22	1	26	1	178	81	19	3	43	5	22	2	91	2
23	1	29	9			20	1	45	2	23	4	94	3
24	3	30	3	Fourballs		21	1	46	1	39	2	96	1
25	1	59	5	KM	TC	22	1	47	3	44	2	97	1
26	1	61	4	43	12	23	1	48	1	48	2	100	1
28	1	62	3	46	6	25	2	49	4	51	4	101	1
30	1	63	5	49	14	26	3	50	1	57	4	103	1
31	1	64	5	50	6	28	4	51	3	62	4		
35	1	65	8	52	12	29	3	52	3	66	1		
37	2	66	11	53	8	30	3	53	2	68	1		
40	2	67	19	54	8	31	2	54	2	69	4		
42	1	68	6	55	10	32	6	55	7	70	4		
				56	2	33	1	56	3	71	1		
				59	10	34	3	57	5	73	1		
				60	4	35	1	59	2	75	1		
				67	4	36	4	60	5	77	6		
						37	6	62	2	78	2		
						38	9	64	1	84	3		
						39	3	67	1	85	2		

TEST SUITE REDUCTION

Another way of testing the second-order mutation technique:

- Puts into the reduced suite the test case that kills more mutants
- removes the killed mutants
- takes the following test case killing more mutants
- removes the killed mutants

$$T' \text{ if } |T'| \leq |T|$$

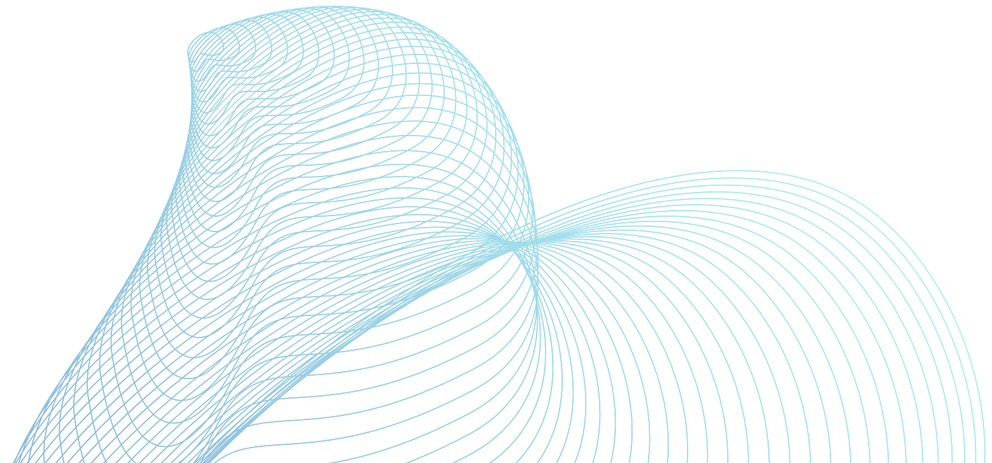
Table VII. Killing matrix for the first-order *Bisect* mutants.

	Test cases																								
Mutants	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
AOIS_1	1	1		1	1		1				1	1		1	1					1					
AOIS_10	1		1		1	1		1	1	1	1		1		1	1		1	1	1	1		1	1	1
AOIS_12	1		1		1	1		1	1	1	1		1		1	1		1	1	1	1		1	1	1
AOIS_13			1		1	1	1	1	1	1	1	1	1		1	1	1	1		1	1	1	1	1	1
AOIS_14	1		1	1	1	1		1	1	1	1		1	1	1	1	1	1	1	1	1		1	1	1
AOIS_15			1		1	1	1	1	1	1	1	1	1		1	1	1	1		1	1	1	1	1	1
AOIS_16	1		1	1	1	1		1	1	1	1		1	1	1	1	1	1	1	1	1		1	1	1
AOIS_17			1		1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1	1
AOIS_18	1		1	1	1	1	1	1	1	1	1		1	1	1	1	1	1	1	1	1	1	1	1	1
AOIS_19	1	1		1	1						1	1		1	1					1					
AOIS_20	1	1		1	1						1	1		1	1						1				
AOIS_25	1	1		1	1		1				1	1		1	1	1				1					
AOIS_27	1	1		1	1						1	1		1	1					1					
AOIS_35		1		1	1						1	1		1	1										
AOIS_37		1		1	1						1	1		1	1										
AOIS_38	1	1		1	1						1	1		1	1										
AOIS_39				1							1	1		1											
AOIS_40				1							1	1		1											
AOIS_41		1		1	1						1	1		1	1										
AOIS_5	1	1	1	1	1		1				1	1	1	1	1					1					
AOIS_52	1	1		1	1	1	1			1	1	1		1	1	1	1			1	1	1			1
AOIS_54	1	1		1	1	1	1			1	1	1		1	1	1	1			1	1	1			1
AOIS_56				1	1						1	1		1	1	1				1					
AOIS_58	1	1		1	1	1	1			1	1	1		1	1	1	1			1	1	1			1
AOIS_71	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIS_72	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIS_75	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIS_76	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIS_77	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIS_78	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIU_11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIU_13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIU_2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
AOIU_6	1				1						1	1		1	1										
AOIU_7	1	1		1	1						1	1		1	1										
AOIU_8	1	1		1	1						1	1		1	1										
AOIU_9	1	1		1	1	1	1			1	1	1		1	1		1			1	1	1			1
AORB_1									1										1					1	
AORB_10					1	1	1			1	1	1		1	1	1	1			1	1	1			1
AORB_3	1	1		1	1	1	1			1	1	1		1	1	1	1			1	1	1			1
AORB_5		1		1	1						1	1		1	1										
AORB_7	1			1	1						1	1		1	1										
ROR_1																			1		1				
ROR_4	1	1		1	1	1	1			1	1	1		1	1	1	1		1	1	1	1			1
Total	31	28	18	35	40	24	23	17	18	24	42	37	18	37	40	25	22	17	18	30	26	20	17	18	24

EXPERIMENT 2

The second experiment demonstrated the **application of second-order mutation to industrial software** classes with available source code and test cases.
The three selected classes were:

- PluginTokenizer from jtopas (web page analysis and parsing system)
- IgnoreList from jester (Java testing tool)
- Ciudad from Tráfico (traffic simulation system with city modeling feature)

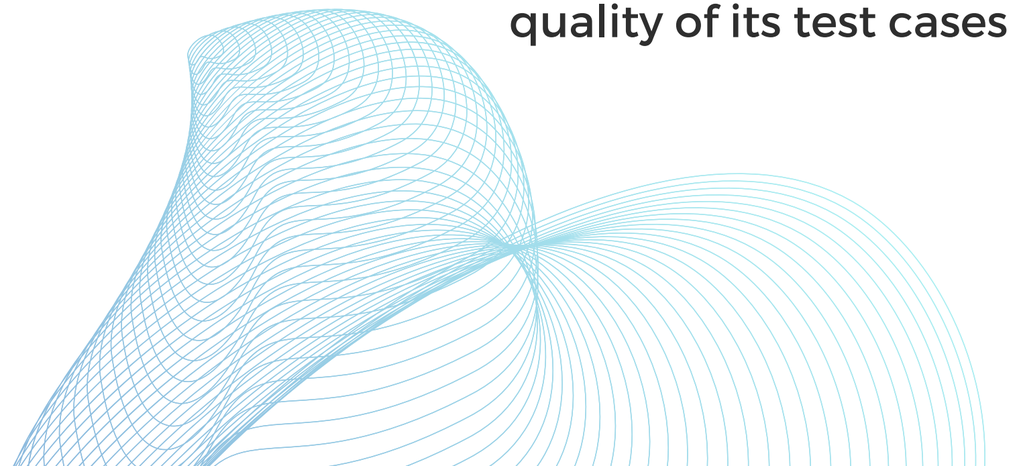


EXPERIMENT 2

Program	No. of sentences	WMC	No. of methods	No. of first-order mutants	No. of available test cases	First-order mutants killed (%)
Ciudad	177	65	23	203	37	92
IgnoreList	26	8	3	27	6	85
PluginTokenizer	157	27	16	94	14	31

- The available test cases from Ciudad and IgnoreList have **good mutation scores**, with 92 and 85%, respectively.

- The Tráfico project was developed using **test driven development**, so the quality of its test cases is, as expected, very high.



EXPERIMENT 2

- For Ciudad, nine of the TCs killed zero mutants, so they could be removed
- All the other TCs managed to kill at least six mutants each
- Therefore, second-order mutation is suitable for the three programs

Ciudad				PluginTokenizer		IgnoreList	
KM	TC	KM	TC	KM	TC	KM	TC
0	9	94	2	18	8	10	1
6	1	95	1	21	1	12	1
15	1	96	3	22	1	19	2
69	1	97	1	25	2	21	1
75	1	98	1	26	1	22	1
86	3	99	1	30	1		
88	1	100	1				
89	1	101	1				
91	1	102	2				
92	2	118	1				
93	1	181	1				

Number of first-order mutants killed (KM) by test cases (TC)

EXPERIMENT 2

	First-order mutants		Second-order mutants		
Program	No.	% Killed	No.	% Reduction	% Killed
Ciudad	203	92	110	54.18	94
IgnoreList	27	85	17	62.96	88
PluginTokenizer	94	31	48	51.06	50

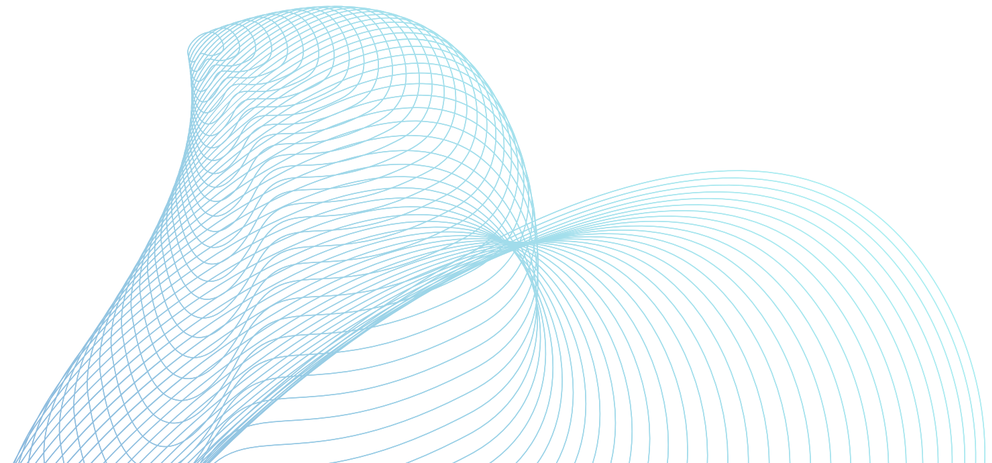
Comparison between first-order and second-order mutations

- The **second-order mutants** were generated using the **DifferentOperators** algorithm to combine the first-order mutants
- The number of mutants reduced by more than half in all cases, and the mutation scores increased as well

COST/RISK ANALYSIS OF K-TH ORDER MUTATION

- Natural question: since second-order mutation is adequate for test quality estimation, can we apply k-th order mutation to increase the savings?

- To answer to this we look back at the previous results



COST/RISK ANALYSIS OF K-TH ORDER MUTATION

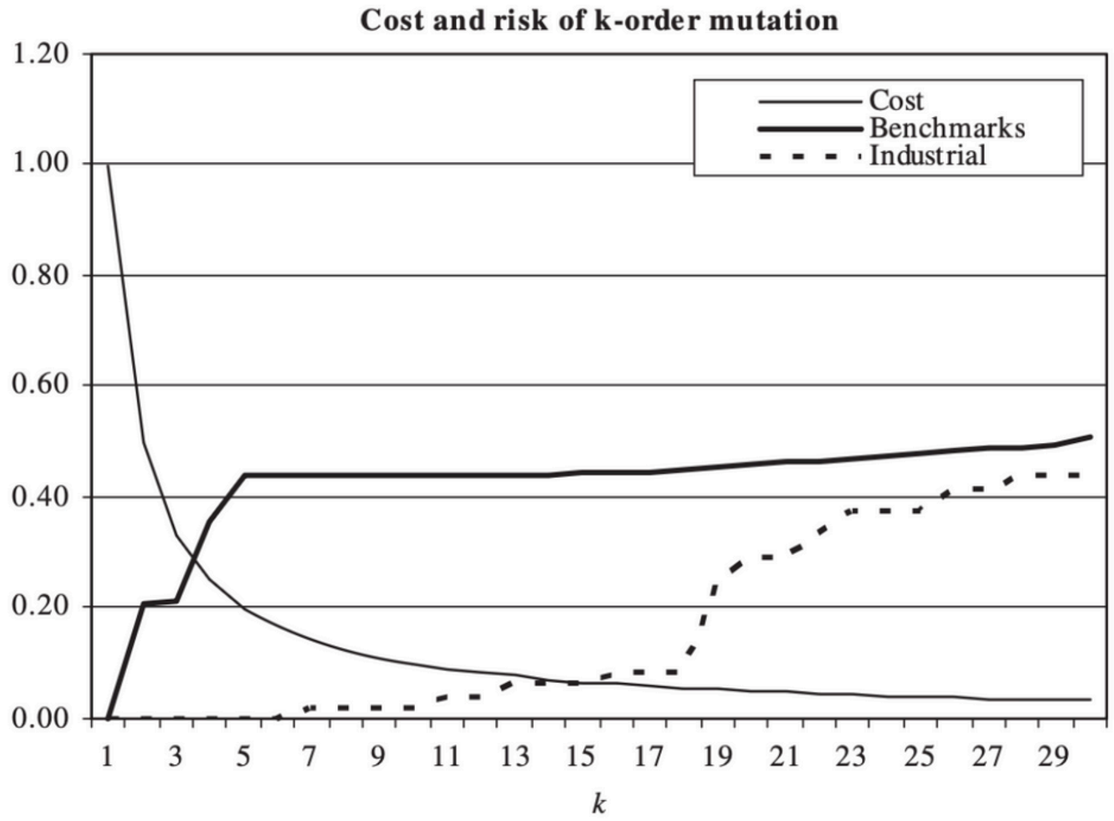
Bisect		Bub		Find		Mid				TriTyp			
KM	TC	KM	TC	KM	TC	KM	TC	KM	TC	KM	TC	KM	TC
17	3	1	174	2	1	11	1	40	3	0	56	86	2
18	5	2	1	3	75	14	4	41	4	3	48	88	1
20	1	25	1	4	20	17	2	42	1	4	48	89	1
22	1	26	1	178	81	19	3	43	5	22	2	91	2
23	1	29	9			20	1	45	2	23	4	94	3
24	3	30	3	Fourballs		21	1	46	1	39	2	96	1
25	1	59	5	KM	TC	22	1	47	3	44	2	97	1
26	1	61	4	43	12	23	1	48	1	48	2	100	1
28	1	62	3	46	6	25	2	49	4	51	4	101	1
30	1	63	5	49	14	26	3	50	1	57	4	103	1
31	1	64	5	50	6	28	4	51	3	62	4		
35	1	65	8	52	12	29	3	52	3	66	1		
37	2	66	11	53	8	30	3	53	2	68	1		
40	2	67	19	54	8	31	2	54	2	69	4		
42	1	68	6	55	10	32	6	55	7	70	4		
				56	2	33	1	56	3	71	1		
				59	10	34	3	57	5	73	1		
				60	4	35	1	59	2	75	1		
				67	4	36	4	60	5	77	6		
						37	6	62	2	78	2		
						38	9	64	1	84	3		
						39	3	67	1	85	2		

Killed mutants by test cases in benchmark programs

- For the Fourballs program, even a 43th-order mutation could be successfully applied;
- For Mid, 11th-order mutation

- For Bub, Find and Trityp, it is a more dangerous situation, since the minimal numbers of mutants killed are 1, 2 and 3, respectively

COST/RISK ANALYSIS OF K-TH ORDER MUTATION



$$risk(k) = 1 - \frac{\text{No. of test cases killing } k \text{ mutants or more}}{\text{total no. of test cases}}$$

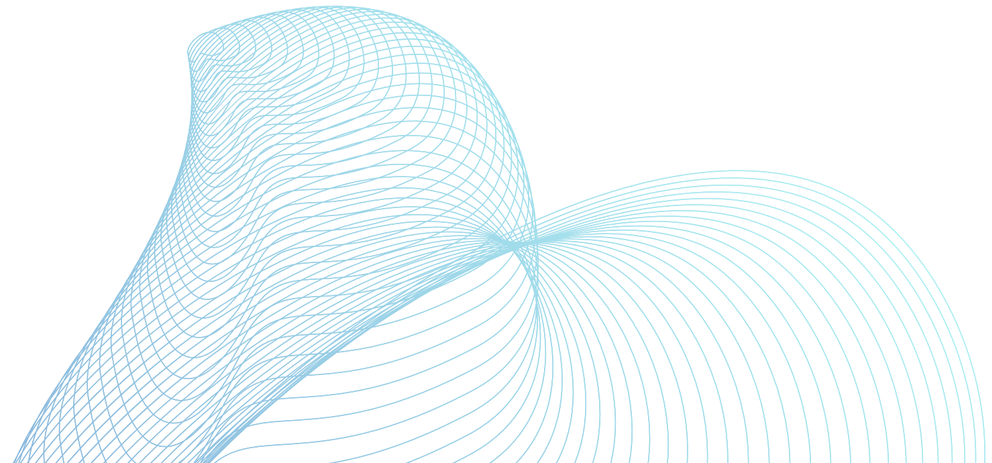
- In the benchmark programs, the risk of second-order mutation is 0.21, while the cost is half the original
- The risk is much lower for industrial software and increases at a slower rate as well
- k-th order mutation should be accepted as a cost-effective testing practice

THREATS TO VALIDITY

How were issues with the validity of the experiments alleviated?

- The first experiment uses benchmark programs that are widely recognized and considered representative by the scientific community.

- The second experiment involves three industrial systems chosen for the availability of their source code and test cases. These systems vary in size and complexity, making them representative of typical industrial applications.



CONCLUSIONS

This article has presented a technique - Combined Mutation - aimed at reducing the cost of mutation testing through the combination of first-order mutants.

Key findings and contributions of this study

- Reduction in Number of Mutants
- Decrease in Equivalent Mutants
- Combination Algorithms