

基于搜索的组合测试

吴化尧

南京大学

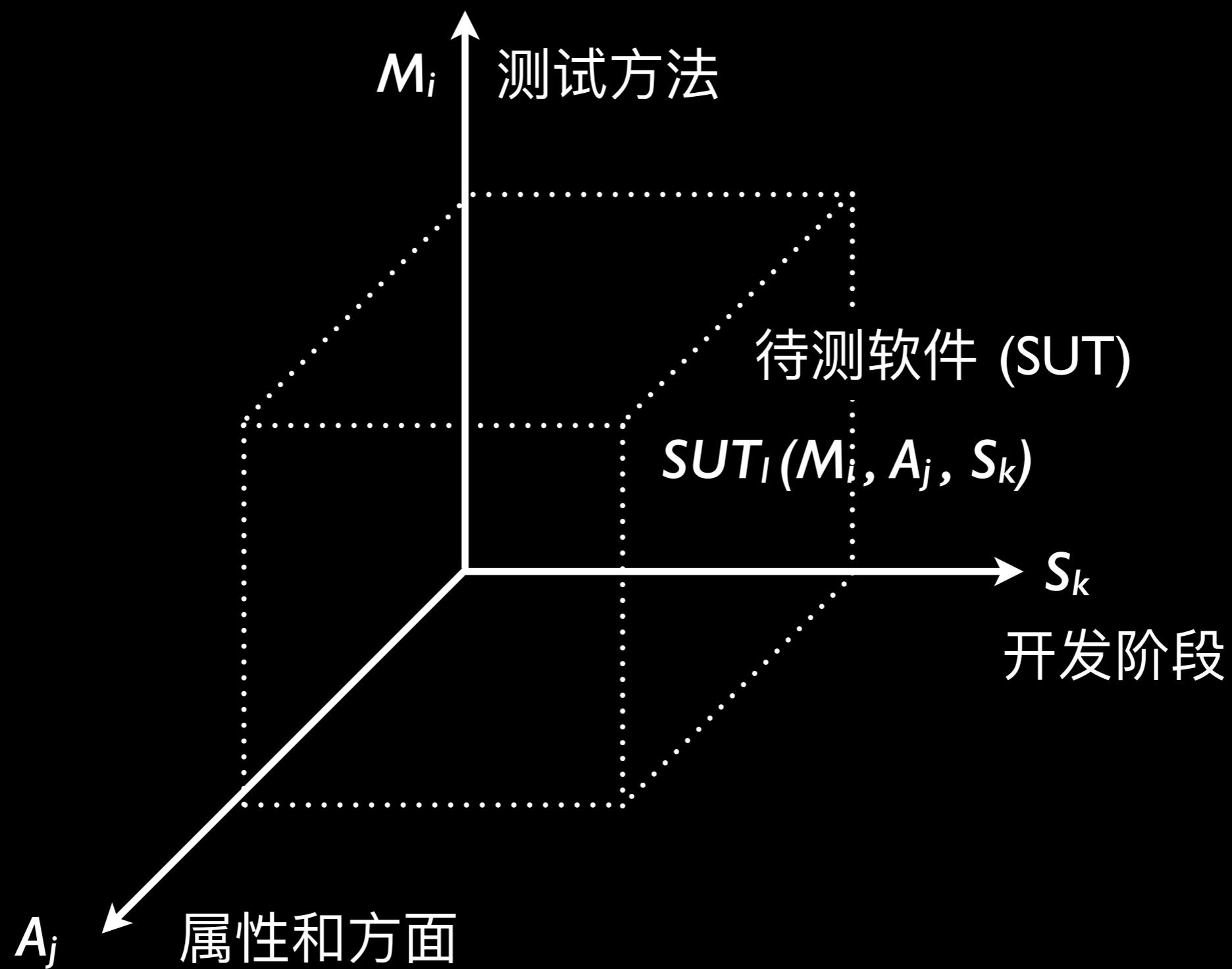
hywu@nju.edu.cn

CSBSE 2018

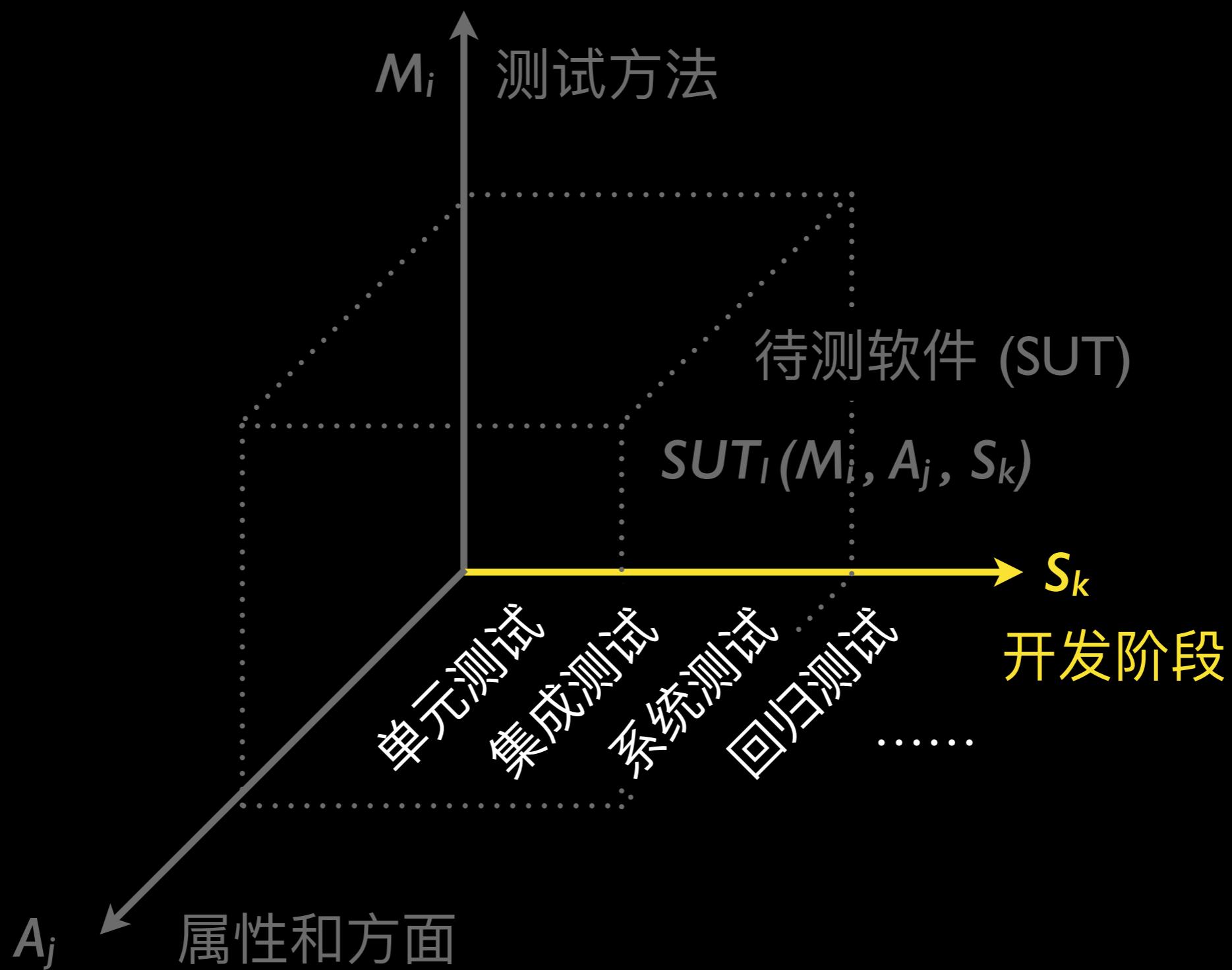
Outline

1. 组合测试 (CT)
2. 基于搜索的组合测试 (SBCT)
 - 基于搜索的测试用例集生成
 - 基于搜索的测试用例优先级排序
 - 组合测试的有效性
3. SBCT 的未来研究

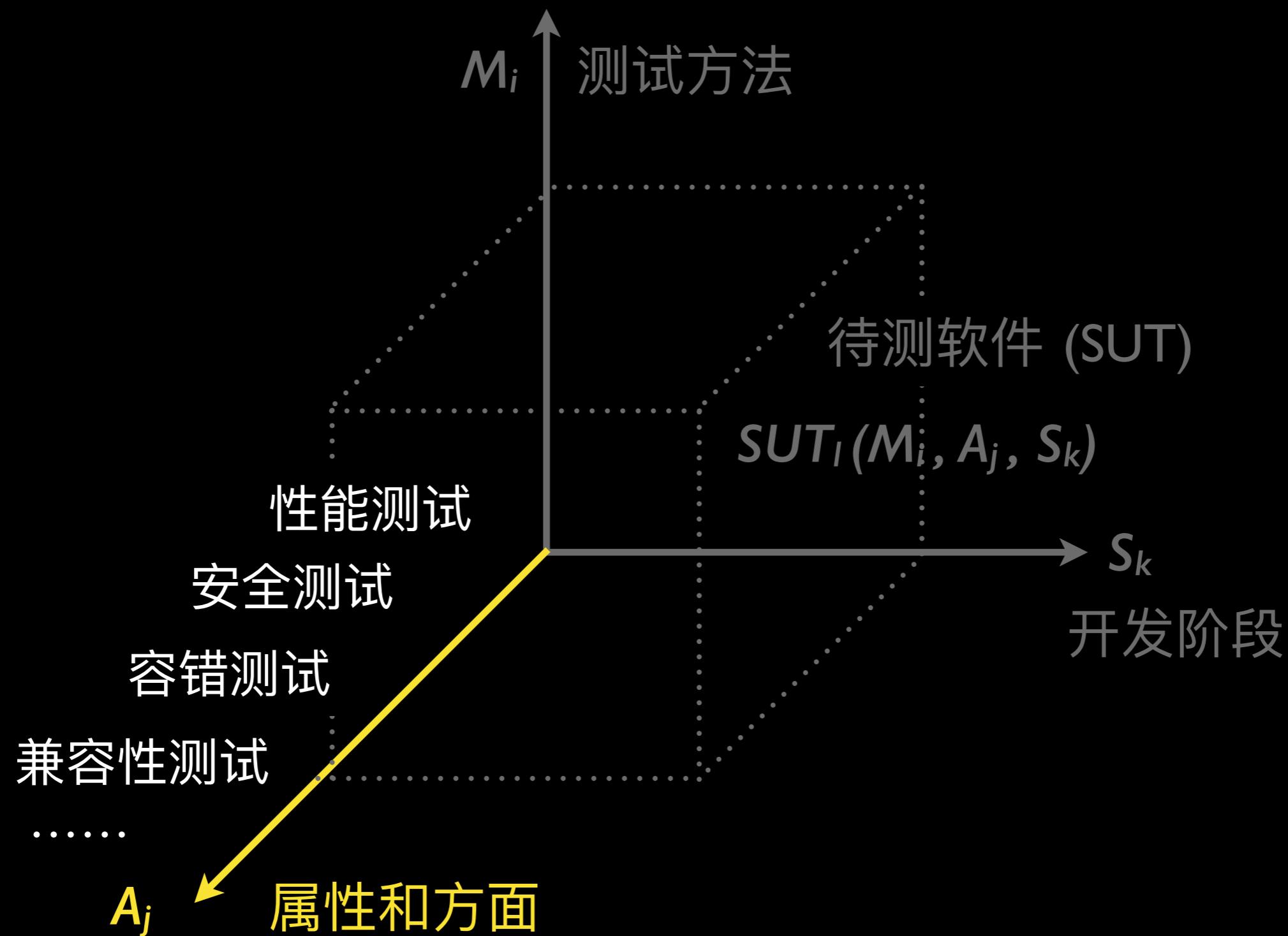
软件测试



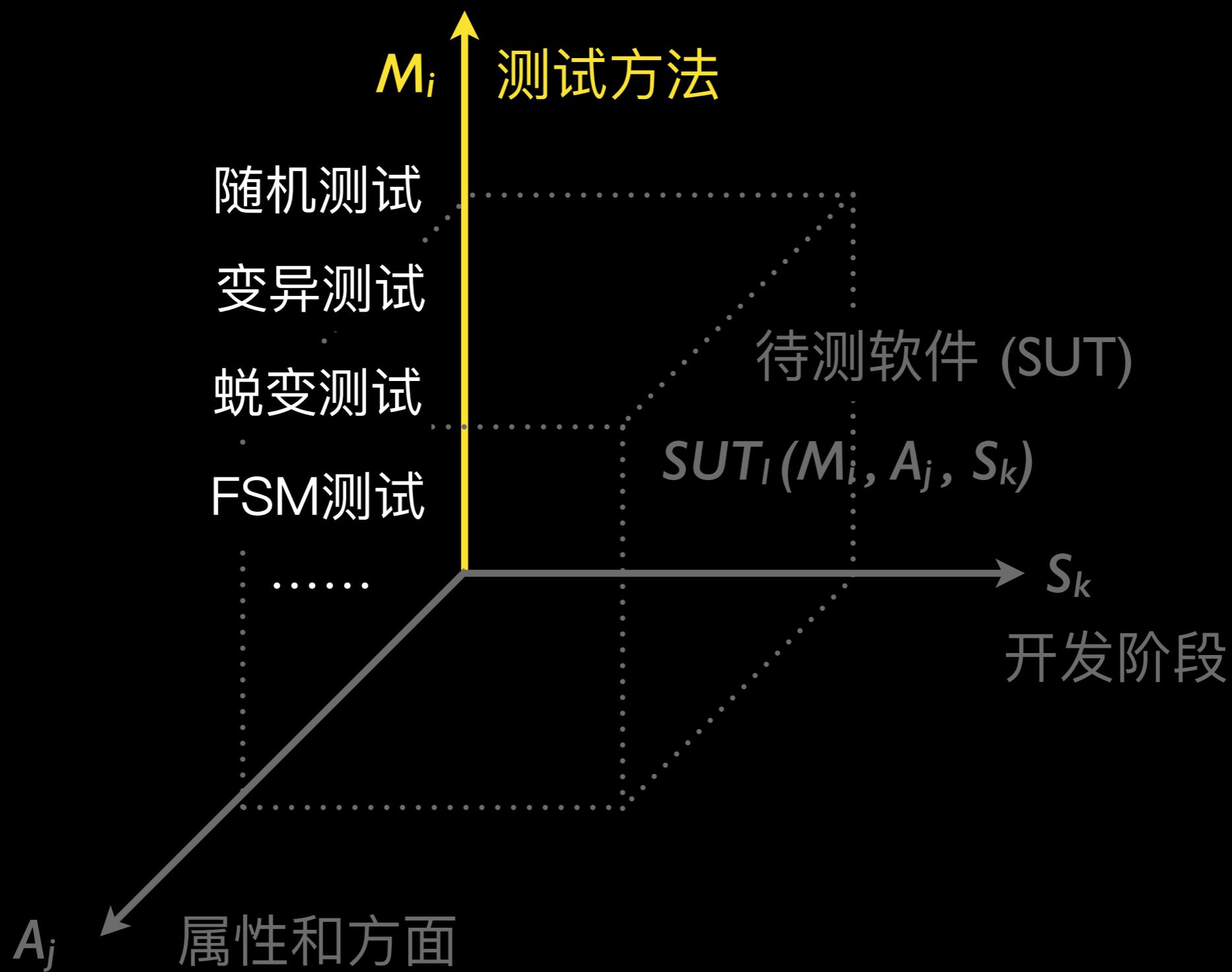
软件测试



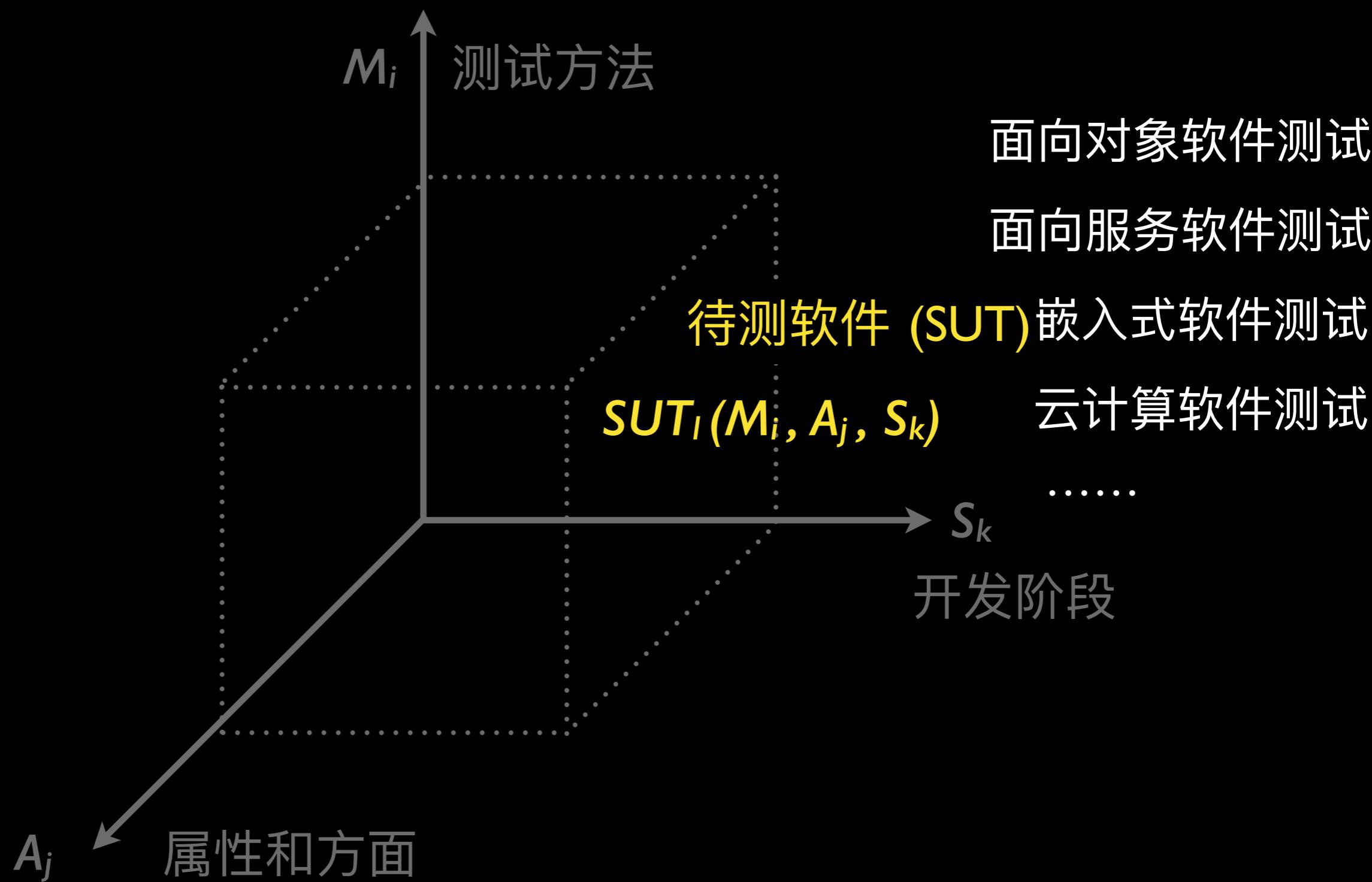
软件测试



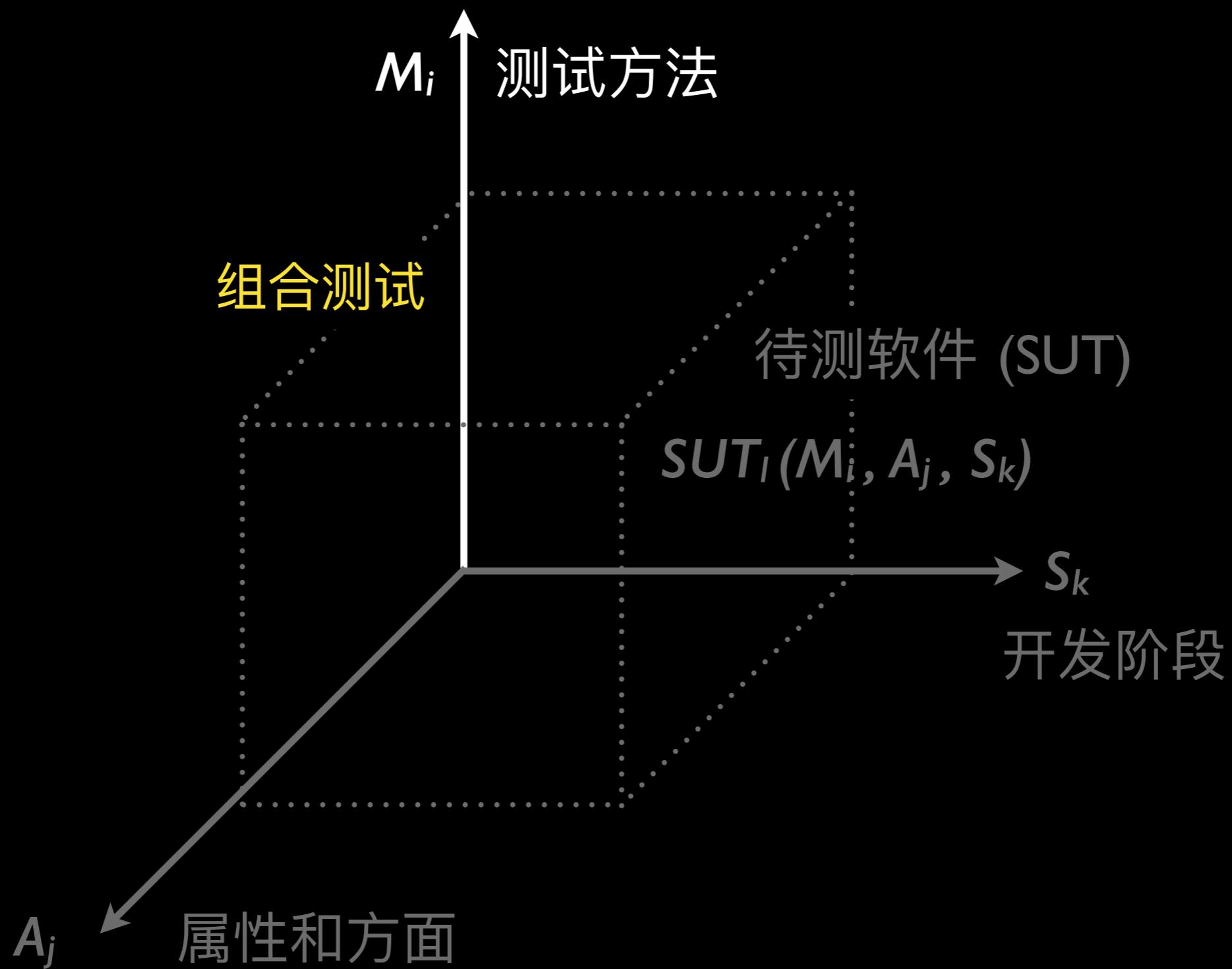
软件测试



软件测试

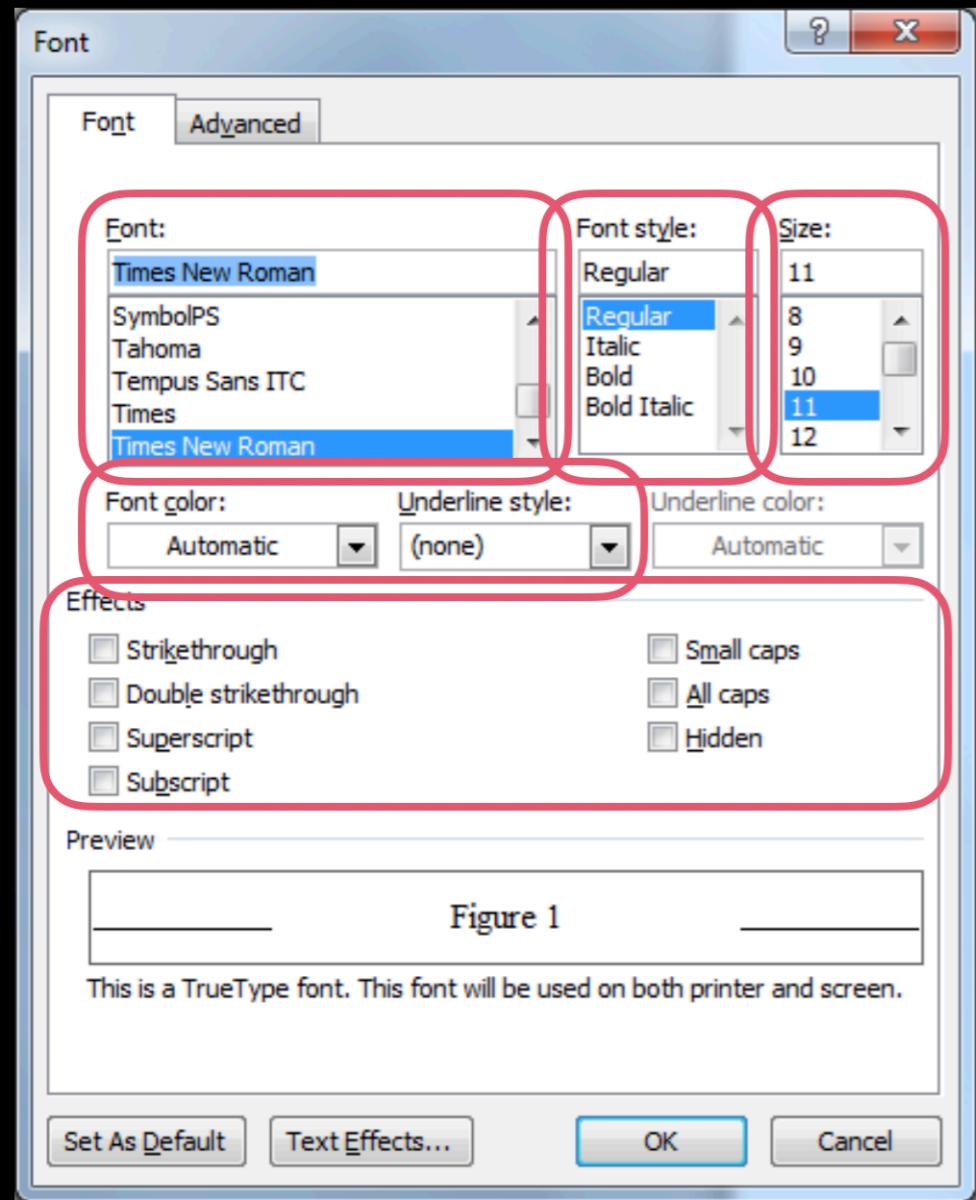


软件测试



为什么要用组合测试？

软件系统的行为通常会受很多参数的影响



40 Fonts
×
4 Styles
×
30 Sizes
×
10 Colors
×
16 Underlines
×
2⁷ Effects
 $\approx 9.8 \times 10^7$ possible inputs

NAME

ls -- list directory contents

SYNOPSIS

ls [-**A****C****F****G****H****L****O****P****R****S****T****U****W**@**a****b**c****d****e****f****g****h****i****k****l****m****n****o****p****q****r****s****t****u****w****x****1**] [**file** ...] *238 × 10¹ inputs***

DESCRIPTION

For each operand that names a **file** of a type other than **directory**, **ls** displays its name as well as any requested, associated information. For each operand that names a **file** of type **directory**, **ls** displays the names of files contained within that **directory**, as well as any requested, associated information.

If no operands are given, the contents of the current **directory** are displayed. If more than one operand is given, non-directory operands are displayed first; **directory** and non-directory operands are sorted separately and in lexicographical order.

The following options are available:

- @ Display extended attribute keys and sizes in long (-l) output.
- 1 (The numeric digit ``one''.) Force output to be one entry per line. This is the default when output is not to a terminal.
- A List all entries except for **.** and **..**. Always set for the super-user.
- a Include directory entries whose names begin with a dot (**.**).
- B Force printing of non-printable characters (as defined by ctype(3) and current locale settings) in file names as **xxx**, where **xxx** is the numeric value of the character in octal.

历史记录

扩展程序

设置

关于

隐私设置

[内容设置...](#)[清除浏览数据...](#)

Google Chrome 浏览器可能会使用网络服务改善您的浏览体验。您可以视情况停用这些服务。 [了解详情](#)

- 使用网络服务帮助解决导航错误
- 在地址栏或应用启动器搜索框中输入搜索字词和网址时，借助联想查询服务自动补齐相关内容
- 预提取资源，以便更快速地加载网页
- 自动向Google报告可能出现的安全事件详情
- 启用针对网上诱骗和恶意软件的防护功能
- 使用网络服务帮助解决拼写错误
- 将使用情况统计信息和崩溃报告自动发送给 Google
- 随浏览流量一起发送“请勿跟踪”请求

$2^{10} \times 5^1 \times 16^1 \times 40^3$ inputs

密码和表单

- 启用自动填充功能，以便点按一次即可填写网络表单。 [管理自动填充设置](#)
- 询问是否保存您在网页上输入的密码。 [管理密码](#)

网络内容

字号：

网页缩放：



什么是组合测试？

问题：对所有的输入组合进行测试往往是不可行的

观察：并不是所有的软件参数都和故障相关

```
def function(a1, a2, a3, a4, ..., a10)

    if a5 < 10:
        # do something

        if a7 > 300:
            # faulty code! (a5 = 0 & a7 = 500)

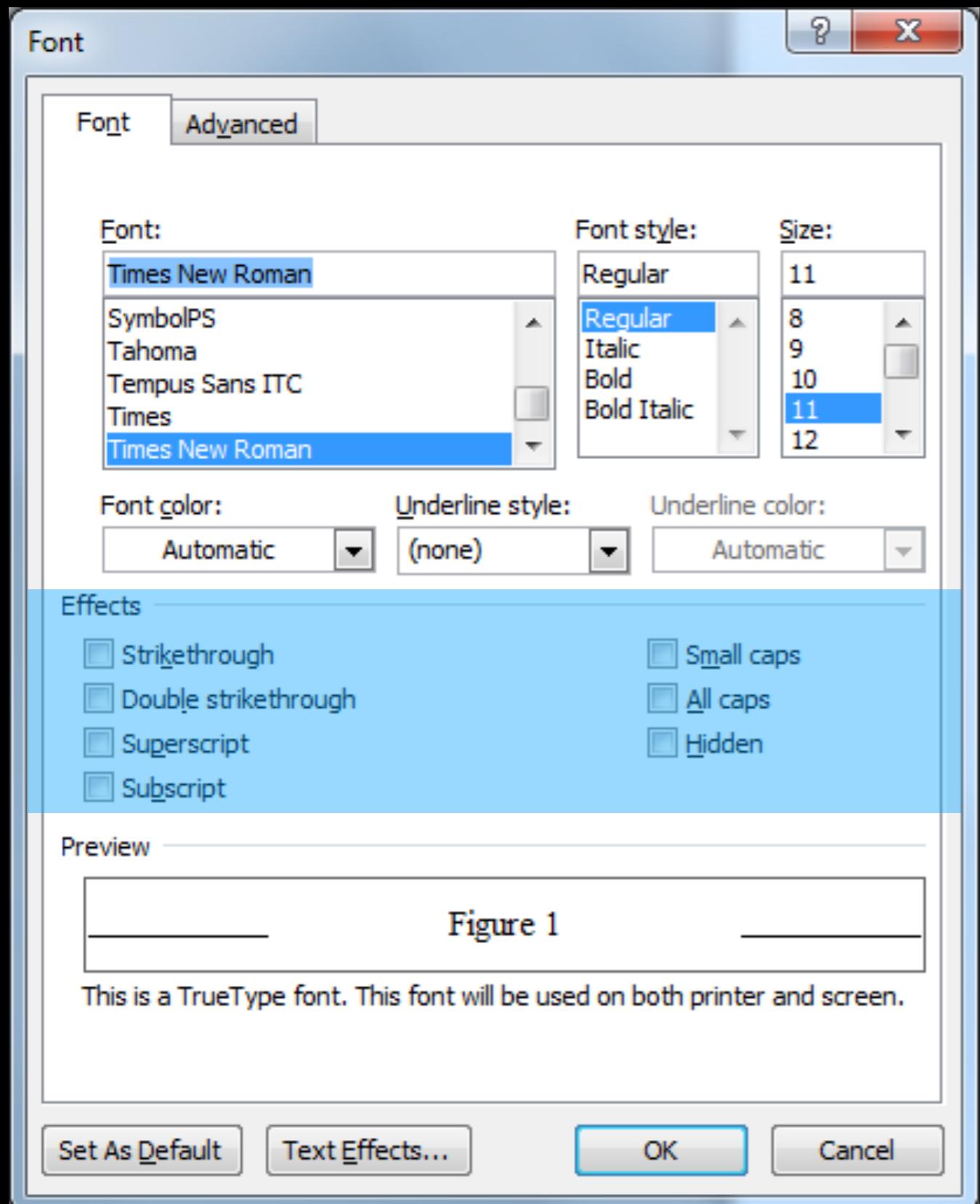
    else:
        # good code, no problem

    # do something else
```

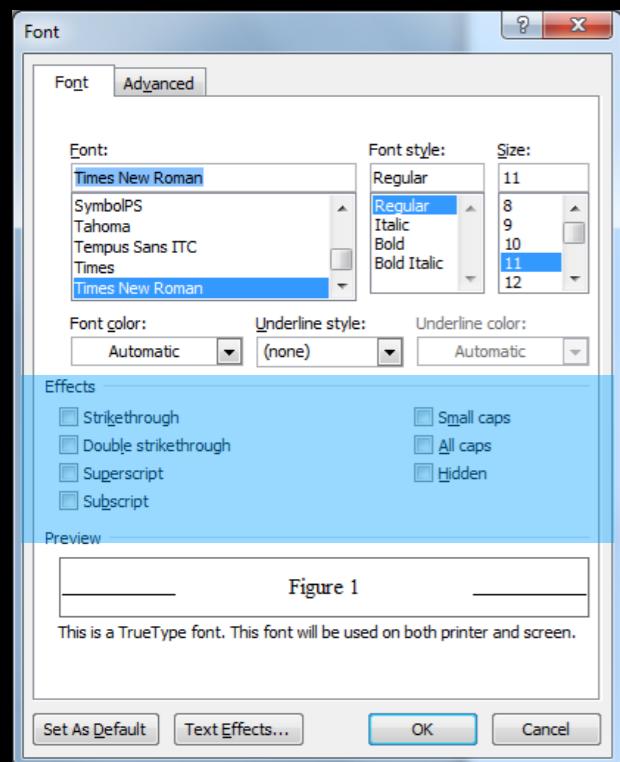
什么是组合测试？

问题：对所有的输入组合进行测试往往是不可行的

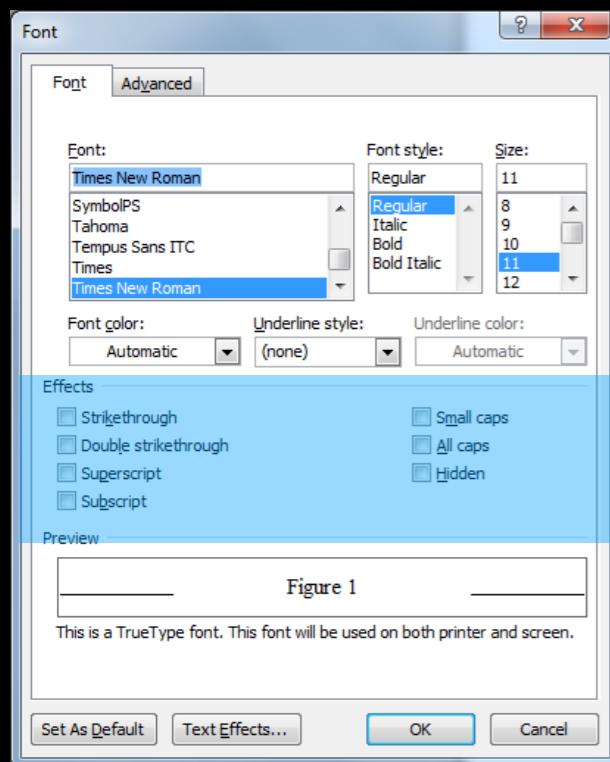
方案：仅需要对特定参数间的相互作用进行测试（测试组合）



Example: Font effect testing

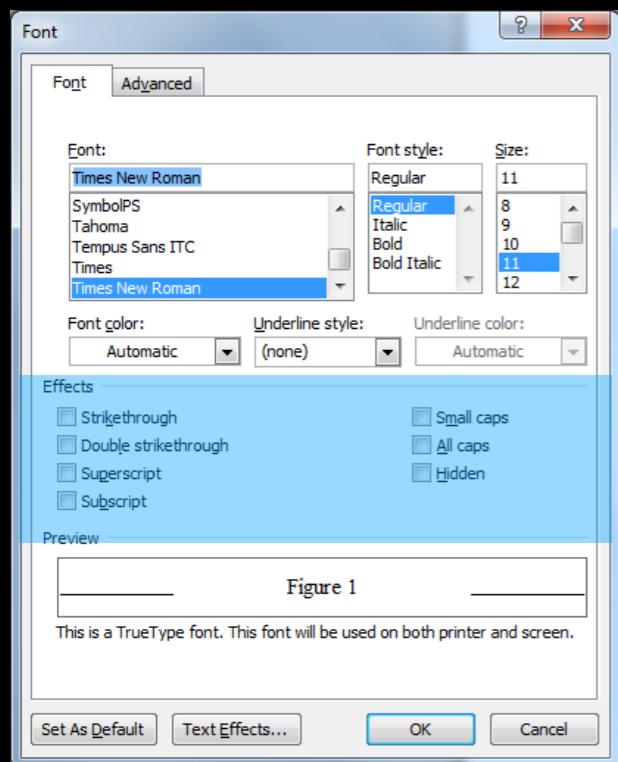


strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	0	0	0	0	0



strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	0	0	0	0	0

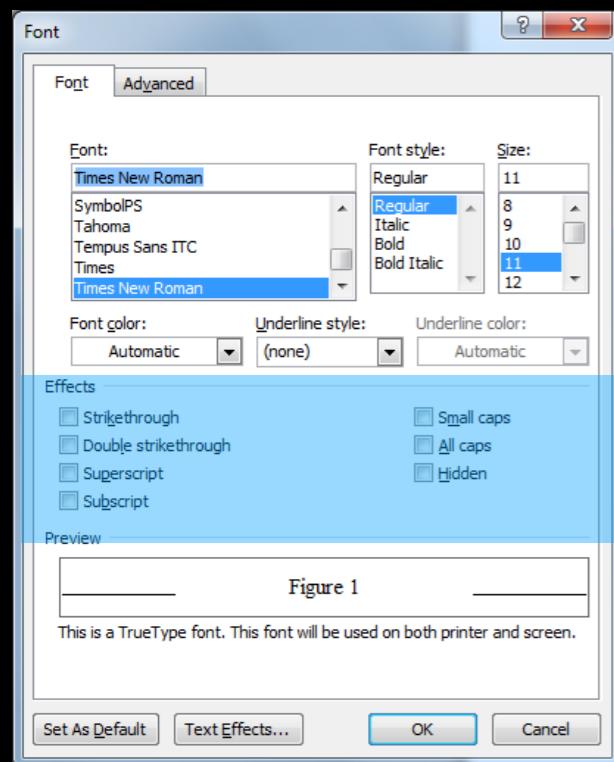
Constraint: *superscript* and *subscript* cannot both be enabled.



strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	0	0	0	0	0	0

为了对所有可能的配置进行穷尽测试，需要 96 条
测试用例

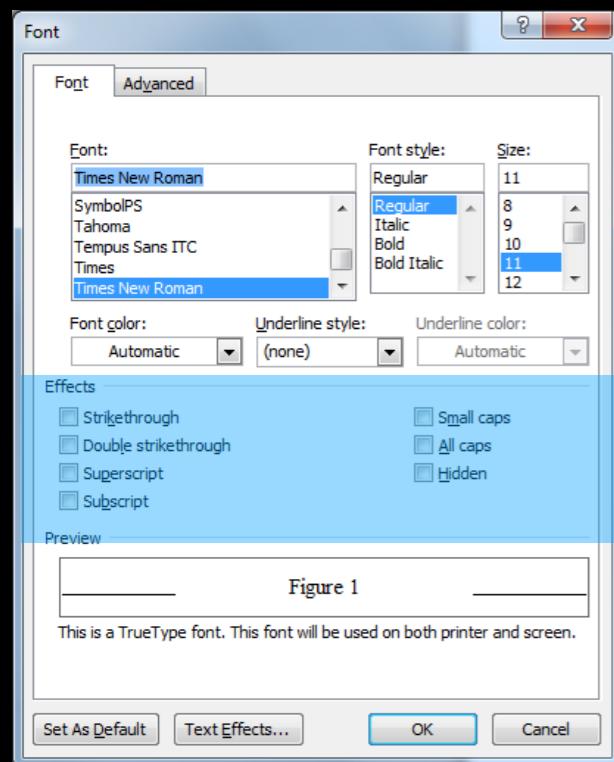
2-way covering array CA(7; 2, 2⁷)



strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0			0			0
	0	0	0	0	0	
		0			0	0
0	0	0		0		
			0	0		
0	0		0	0	0	0
0	0	0				

组合测试使用 *t-way* 覆盖表作为测试用例集，其中任意 *t* 个参数间所有满足约束的取值组合都至少被覆盖一次

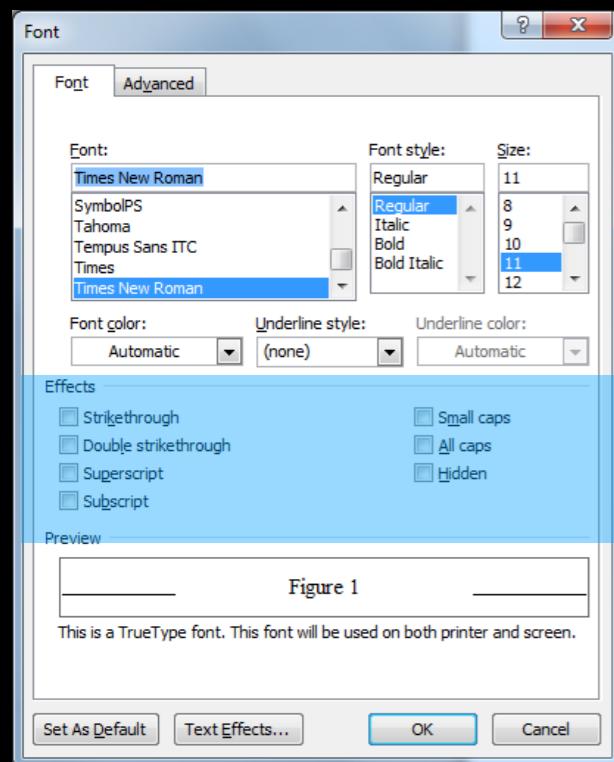
2-way covering array CA(7; 2, 2⁷)



strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0			0			0
	0	0	0	0	0	
		0			0	0
0	0	0		0		
			0	0		
0	0		0	0	0	0
0	0	0				

组合测试使用 *t-way* 覆盖表作为测试用例集，其中任意 *t* 个参数间所有满足约束的取值组合都至少被覆盖一次

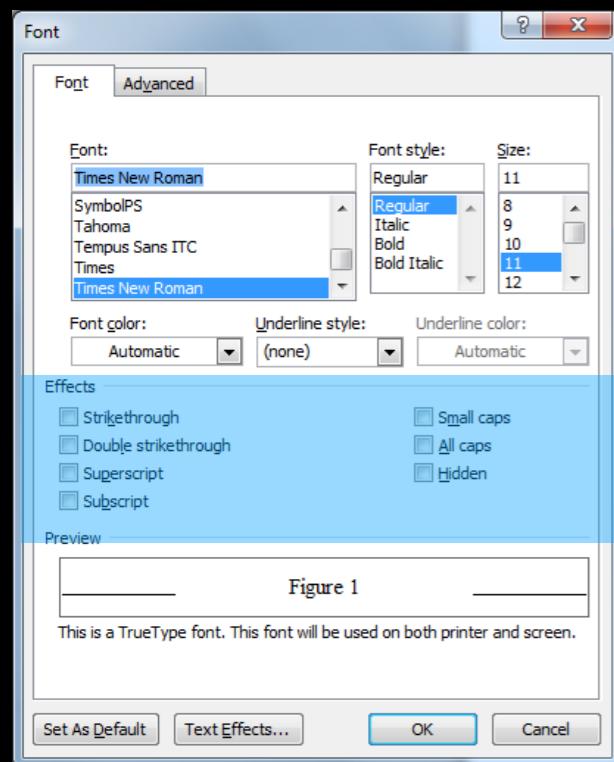
2-way covering array CA(7; 2, 2⁷)



strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	I	I	0	I	I	0
I	0	0	0	0	0	I
I	I	0	I	I	0	0
0	0	0	I	0	I	I
I	I	I	0	0	I	I
0	0	I	0	0	0	0
0	0	0	I	I	I	I

组合测试使用 *t-way* 覆盖表作为测试用例集，其中任意 *t* 个参数间所有满足约束的取值组合都至少被覆盖一次

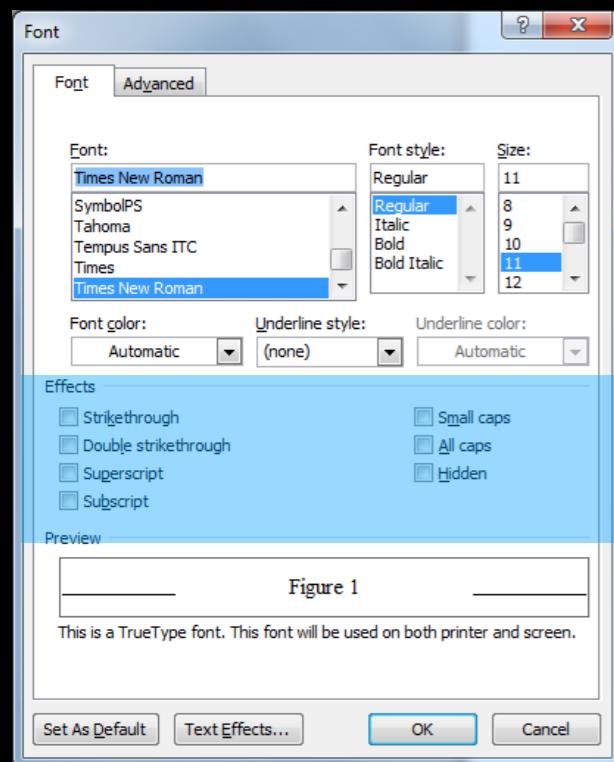
2-way covering array CA(7; 2, 2⁷)



striketh rough	double striketh rough	super script	sub script	small caps	all caps	hidden
0	I	I	0	I	I	0
I	0	0	0	0	0	I
I	I	0	I	I	0	0
0	0	0	I	0	I	I
I	I	I	0	0	I	I
0	0	I	0	0	0	0
0	0	0	I	I	I	I

组合测试使用 *t-way* 覆盖表作为测试用例集，其中任意 *t* 个参数间所有满足约束的取值组合都至少被覆盖一次

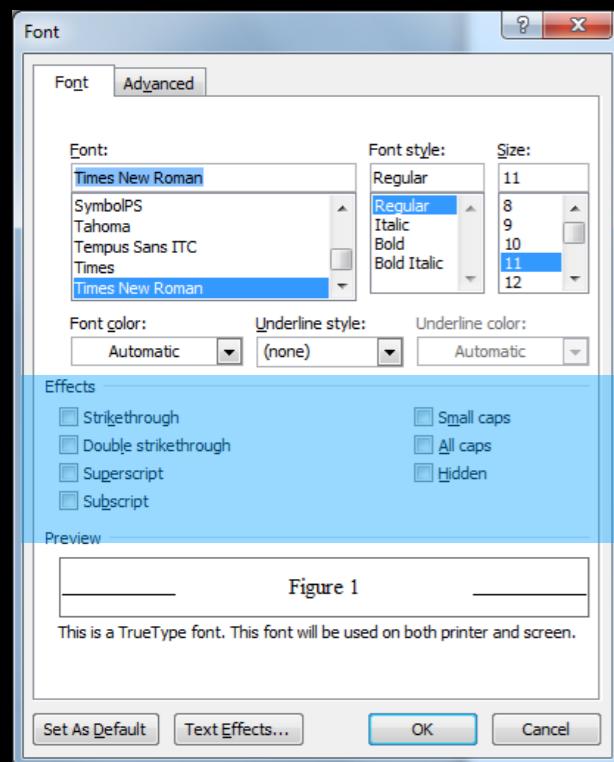
2-way covering array CA(7; 2, 2⁷)



strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0			0			0
	0	0	0	0	0	
		0			0	0
0	0	0		0		
			0	0		
0	0		0	0	0	0
0	0	0				

组合测试使用 *t-way* 覆盖表作为测试用例集，其中任意 *t* 个参数间所有满足约束的取值组合都至少被覆盖一次

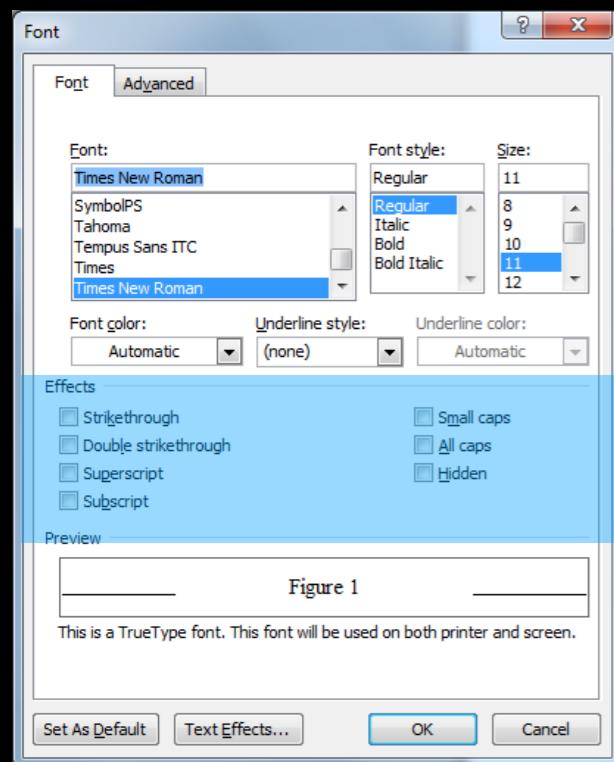
2-way covering array CA(7; 2, 2⁷)



strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0			0			0
	0	0	0	0	0	
		0			0	0
0	0	0		0		
			0	0		
0	0		0	0	0	0
0	0	0				

组合测试使用 *t-way* 覆盖表作为测试用例集，其中任意 *t* 个参数间所有满足约束的取值组合都至少被覆盖一次

2-way covering array $CA(7; 2, 2^7)$



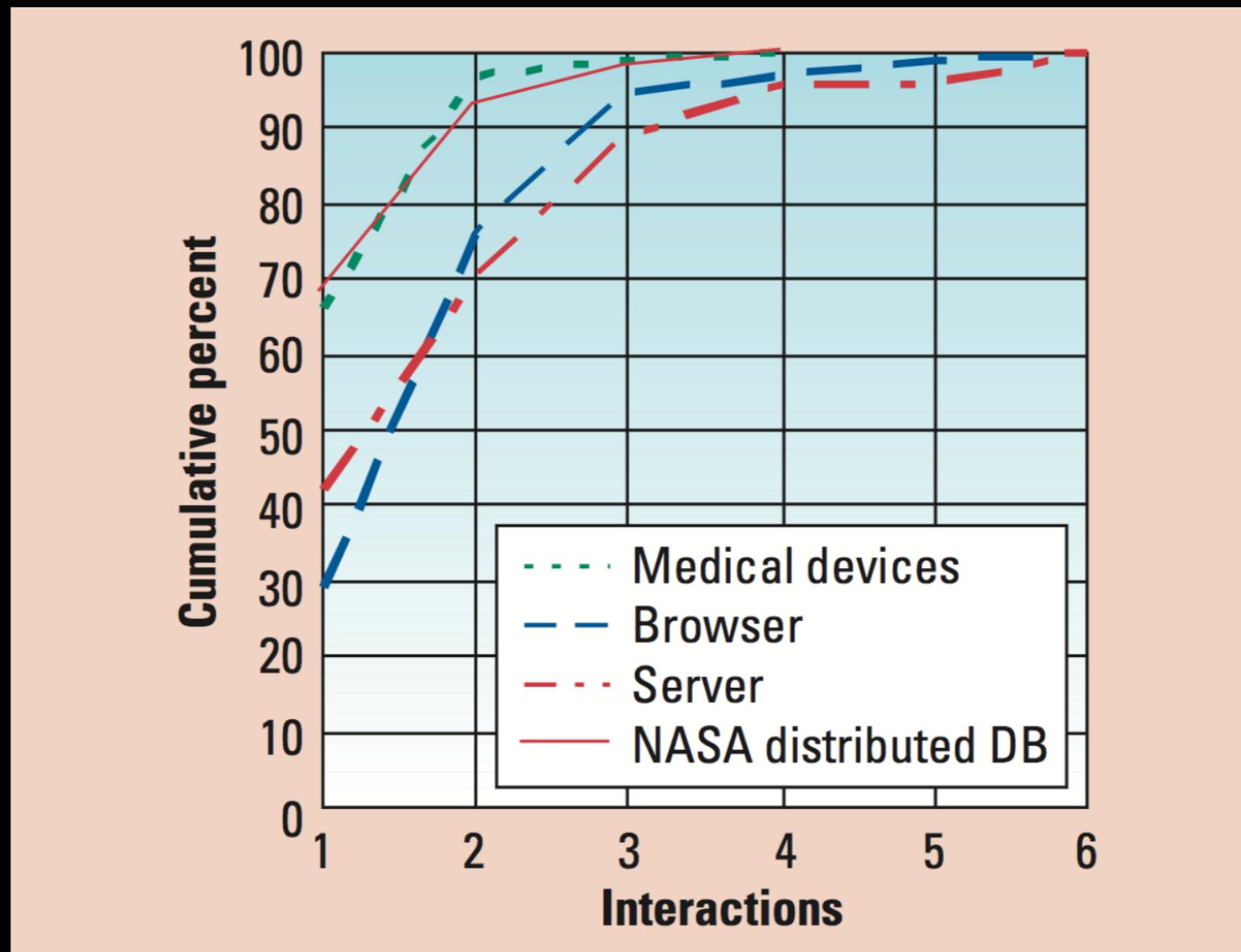
*fault causing
combination*

strikethrough	double strikethrough	super script	sub script	small caps	all caps	hidden
0	I	I	0	I	I	0
I	0	0	0	0	0	I
I	I	0	I	I	0	0
0	0	0	I	0	I	I
I	I	I	0	0	I	I
0	0	I	0	0	0	0
0	0	0	I	I	I	I

如果与软件故障相关的参数个数不超过 t 的话，那么 t -way 组合测试在故障检测能力上与穷尽测试等效

为什么组合测试有效？

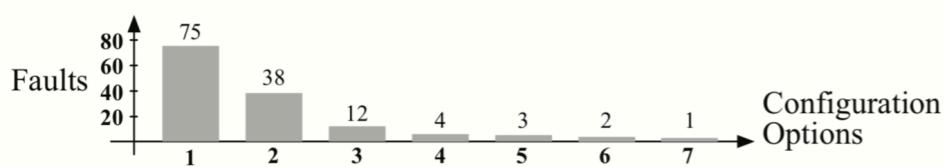
大部分软件故障都是由一个参数或两个参数间的相互作用所引起的，与软件故障相关的参数个数不太可能超过六 [Kuhn 2002]



为什么组合测试有效？

近期在开源软件上的一些研究也进一步证实了这一发现

Some configuration options enabled	78 (58%)
a	59
$a \wedge b$	13
$a \wedge b \wedge c$	5
$a \wedge b \wedge c \wedge d \wedge e$	1
Some configuration options disabled	27 (20%)
$\neg a$	16
$\neg a \wedge \neg b$	8
$\neg a \wedge \neg b \wedge \neg c$	1
$\neg a \wedge \neg b \wedge \neg c \wedge \neg d$	1
$\neg a \wedge \neg b \wedge \neg c \wedge \neg d \wedge \neg e \wedge \neg f \wedge \neg g$	1
Some options enabled and some disabled	30 (22%)
$(\neg a \wedge b) \vee (a \wedge \neg b)$	17
$(a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c)$	6
$(a \wedge b \wedge \neg c \wedge \neg d) \vee (a \wedge b \wedge c \wedge \neg d)$	3
$(a \wedge b \wedge c \wedge d \wedge \neg e) \vee (\neg a \wedge \neg b \wedge \neg c \wedge \neg d \wedge e)$	2
$a \wedge \neg b \wedge \neg c \wedge \neg d \wedge \neg e \wedge \neg f$	1
$a \wedge b \wedge \neg c \wedge \neg d \wedge \neg e \wedge \neg f$	1



24 open source software (135 faults)

84% of faults are caused by 1 or 2 parameters

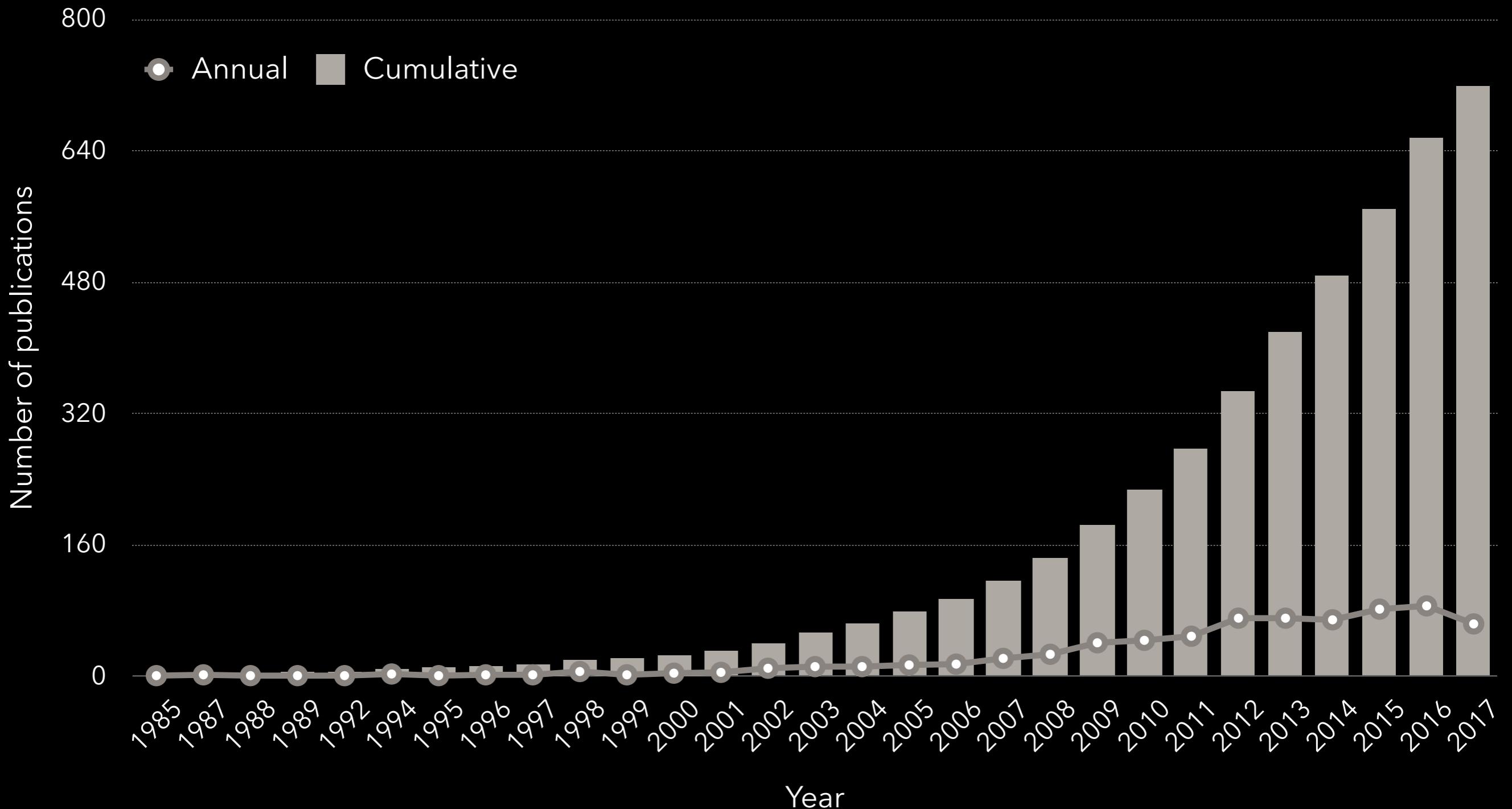
[Medeiros 2016]

Drupal (160 faults)

83% of faults are caused by 2 parameters

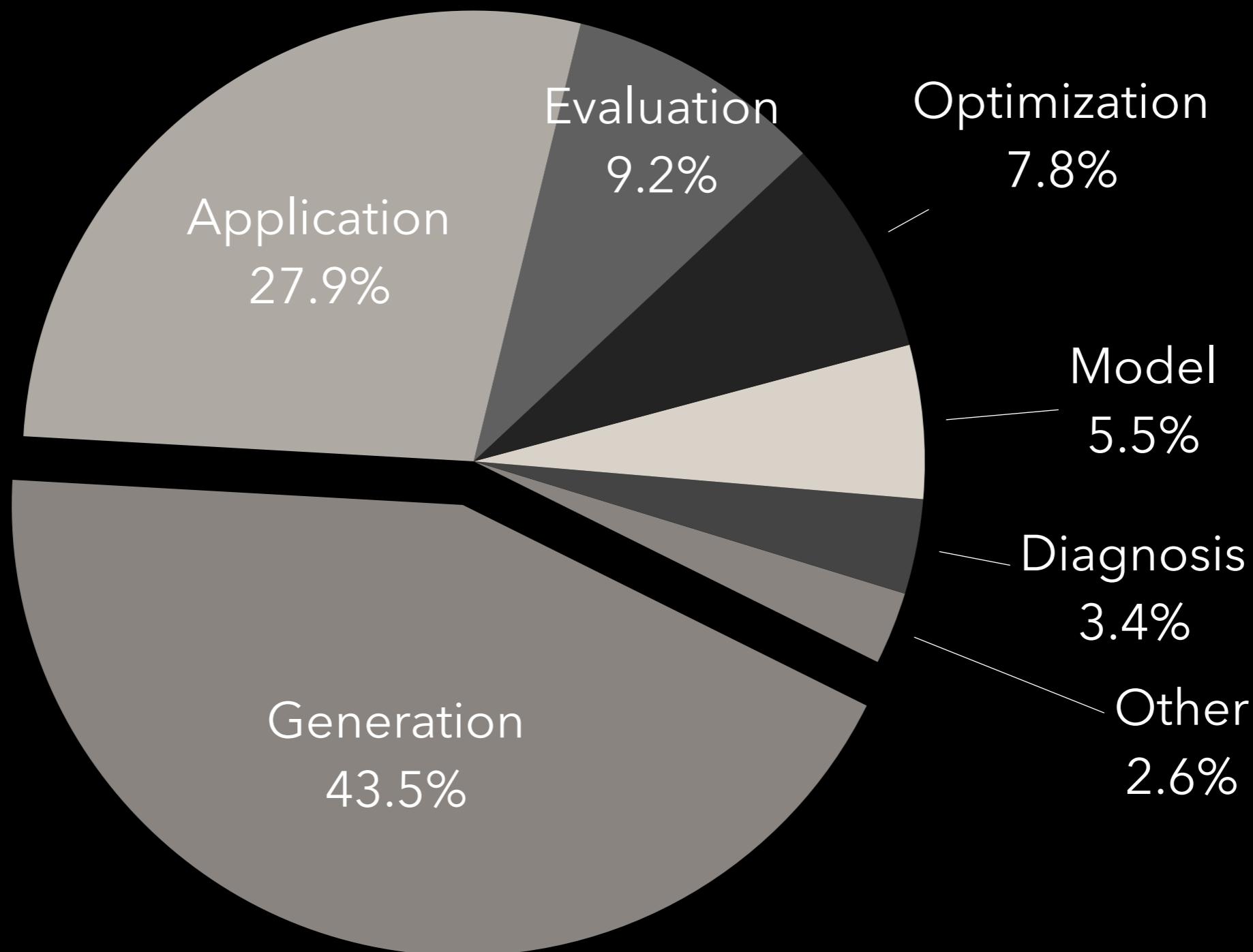
[Sanchez 2017]

研究现状 研究论文



Source: combinatorial testing repository, August 2018

研究现状 研究领域



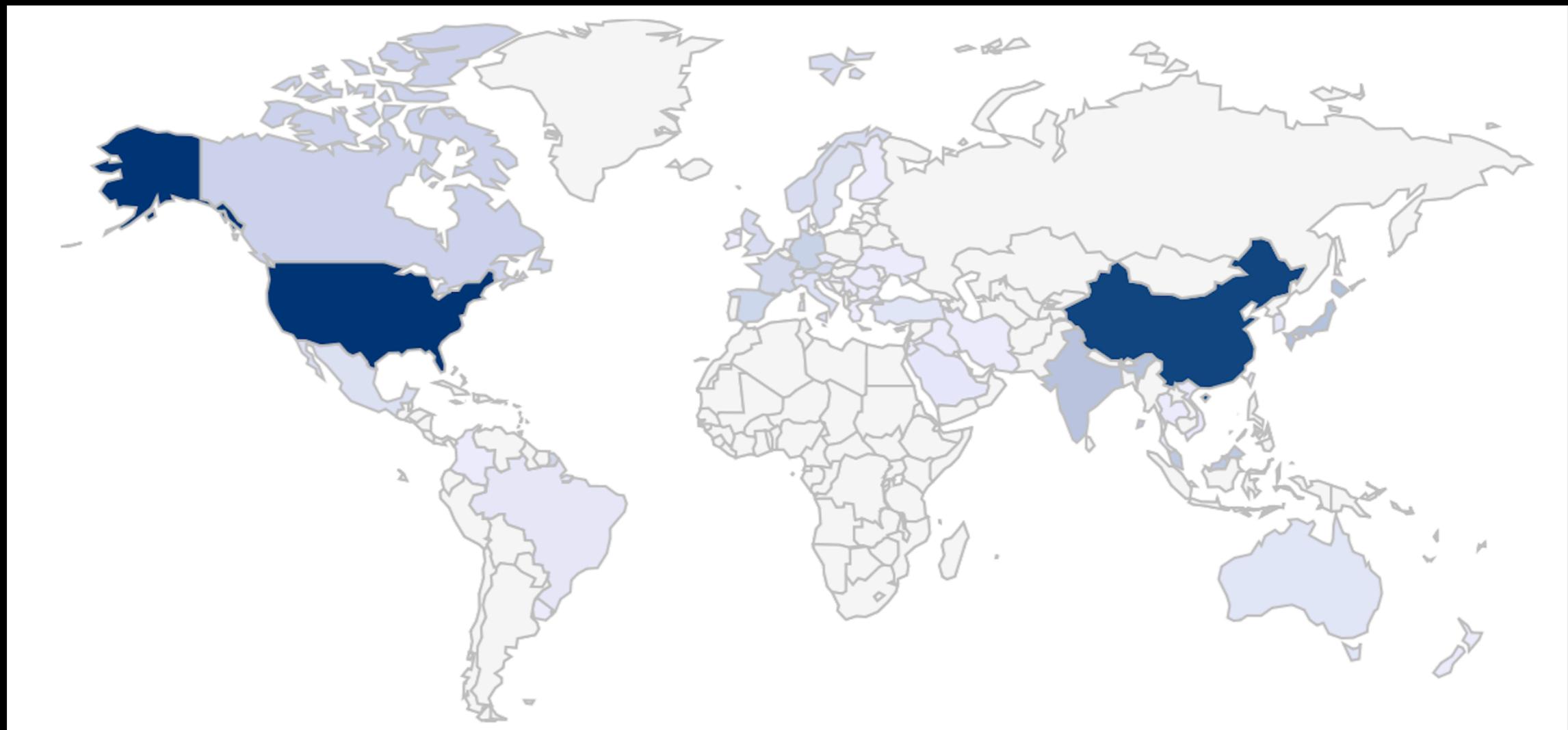
Source: combinatorial testing repository, August 2018

研究现状 研究者和研究机构

840 researchers

45 countries

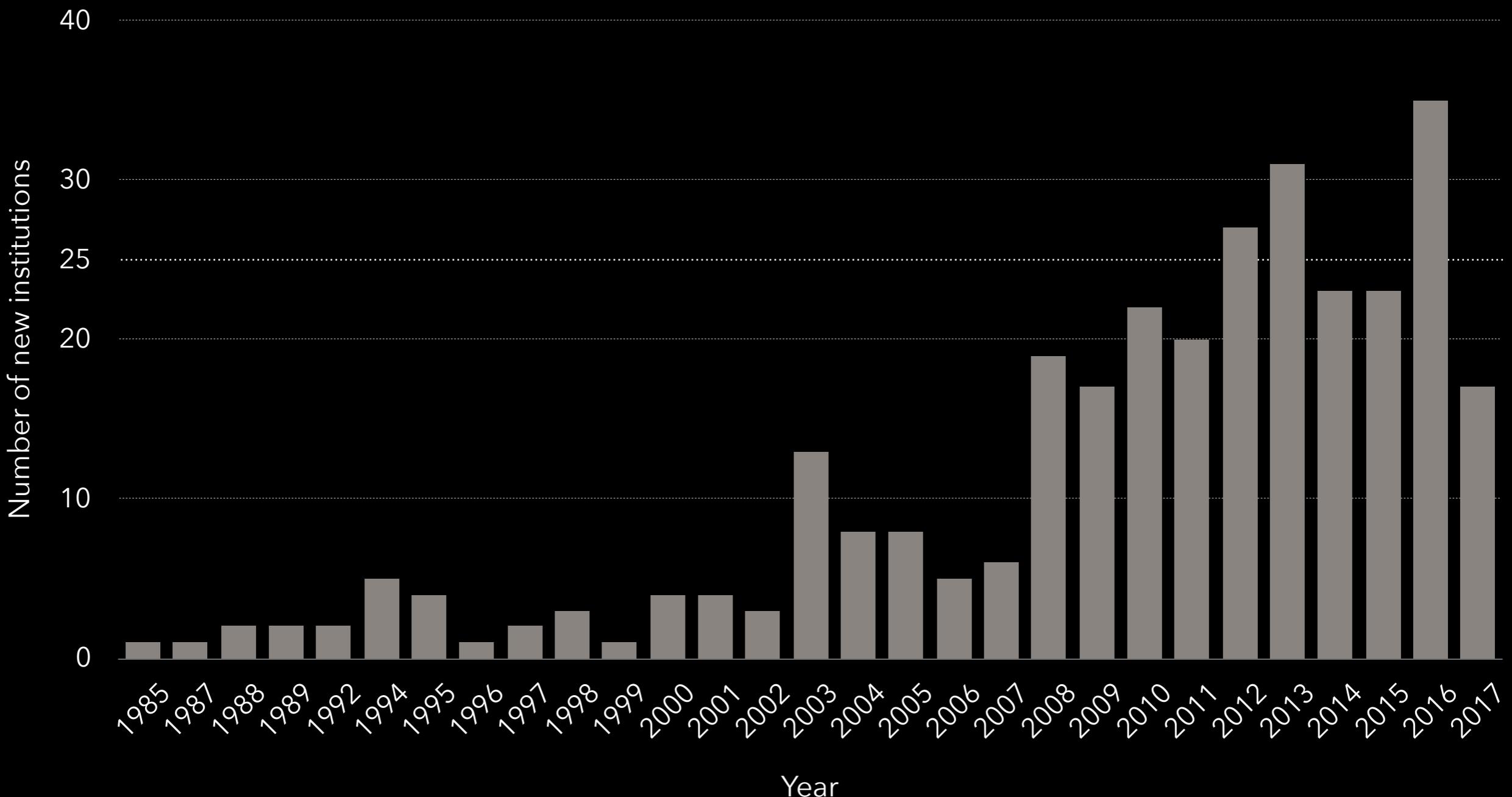
300 institutions (academia: 214 & industry: 86)



Source: combinatorial testing repository, August 2018

研究现状 研究者和研究机构

近五年，平均每年有 25 个新的研究机构首次开展组合测试研究



Source: combinatorial testing repository, August 2018

研究现状 工业界及应用

Configuration Testing

Input Parameter Testing

GUI Testing

Software Product Line Testing

Concurrent Program Testing

Web Service Testing

Security Testing

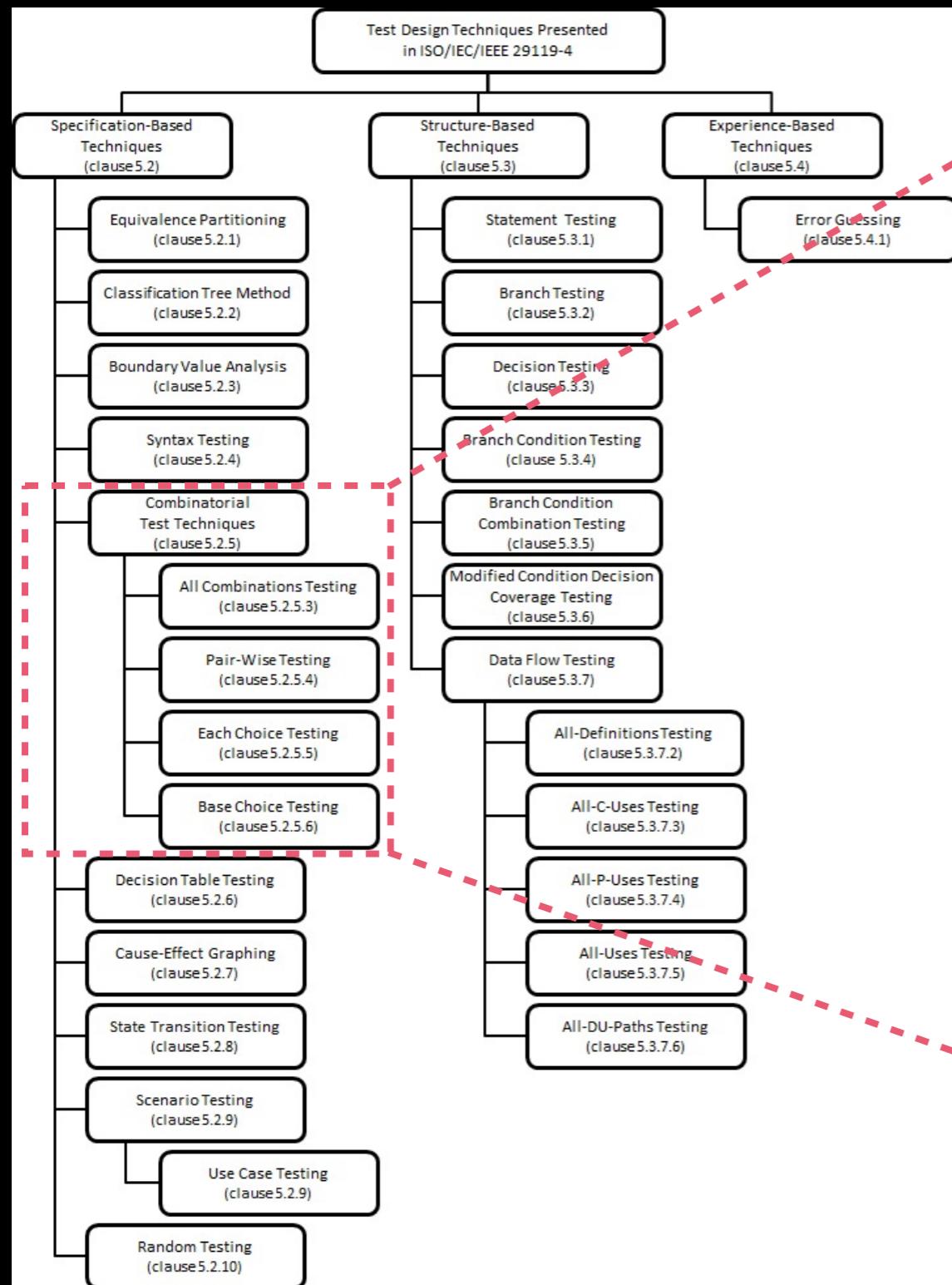
App Testing

Big Data Application Testing

...



研究现状 ISO/IEC/IEEE 29119 软件测试标准



Combinatorial Test Techniques (clause 5.2.5)

All Combinations Testing (clause 5.2.5.3)

Pair-wise Testing (clause 5.2.5.4)

Each Choice Testing (clause 5.2.5.5)

Base Choice Testing (clause 5.2.5.6)

基于搜索的组合测试

使用启发式搜索技术来解决组合测试中的各类问题

$$\begin{array}{ccc} \text{Search} & \text{CT} & \text{Search Based} \\ \text{Techniques} & + & \text{Problems} \\ & & = \quad \text{Combinatorial Testing} \\ & & \quad \quad \quad (\text{SBCT}) \end{array}$$

基于搜索的组合测试

使用启发式搜索技术来解决组合测试中的各类问题

$$\begin{matrix} \text{Search} \\ \text{Techniques} \end{matrix} + \text{Modelling} = \begin{matrix} \text{Search Based} \\ \text{Modelling} \end{matrix}$$

基于搜索的组合测试

使用启发式搜索技术来解决组合测试中的各类问题

$$\begin{matrix} \text{Search} \\ \text{Techniques} \end{matrix} + \text{Generation} = \begin{matrix} \text{Search Based} \\ \text{Covering Array Generation} \end{matrix}$$

基于搜索的组合测试

使用启发式搜索技术来解决组合测试中的各类问题

$$\text{Search Techniques} + \text{Optimisation} = \text{Search Based Test Suite Optimisation}$$

基于搜索的组合测试

使用高效的启发式搜索技术来解决组合测试中的各类问题

1. 组合测试用例生成问题
2. 组合测试用例优先级排序问题
3. 组合测试有效性问题

基于搜索的组合测试

使用高效的启发式搜索技术来解决组合测试中的各类问题

1. 组合测试用例生成问题
2. 组合测试用例优先级排序问题
3. 组合测试有效性问题

覆盖表生成

组合测试中最为核心且最受关注的问题

- 为给定的测试模型生成规模尽可能小的 t -way covering array
- 在某些特殊情况下我们能构造理论最优的覆盖表（例如正交表），但对任意的待测系统，目前仍不知道其对应最小覆盖表的规模

数学方法

能快速生成规模最小的覆盖表
仅适用于满足特殊性质的测试模型
难于对约束等特殊情况进行处理

计算方法

适用于任何测试模型
通常需要一定的计算时间

计算方法

适用于任何测试模型
通常需要一定的计算时间

One-Test-at-a-Time

每次生成一条测试用例

0	0	0	0
0	0	0	0
I	I	0	0
0	0	0	0
I	I	0	0
0	I	I	0
0	0	0	0
I	I	0	0
0	I	I	0
0	I	0	I

In-Parameter-Order

依次对参数进行水平和垂直扩展

0	0		
0	I		
I	0		
I	I		
0	0	0	0
0	I	0	0
I	0	I	0
I	I	0	0
0	I	I	0
0	I	0	I
0	0	0	0
I	I	0	0
0	I	I	0
0	I	0	I
I	0	I	I

Evolve-Test-Set

依次修改表中的某些位置

0	0	I	I
0	I	I	I
0	0	I	I
0	0	0	I
I	I	I	0
0	0	I	0
0	I	0	0
0	0	I	0
0	0	0	I
I	I	I	I
0	0	0	0
I	I	0	0
0	I	I	0
0	I	0	I
I	0	I	I

基于搜索的覆盖表生成

计算方法框架

One-Test-at-a-Time

In-Parameter-Order

Evolve-Test-Set



搜索算法

Hill Climbing

Simulated Annealing

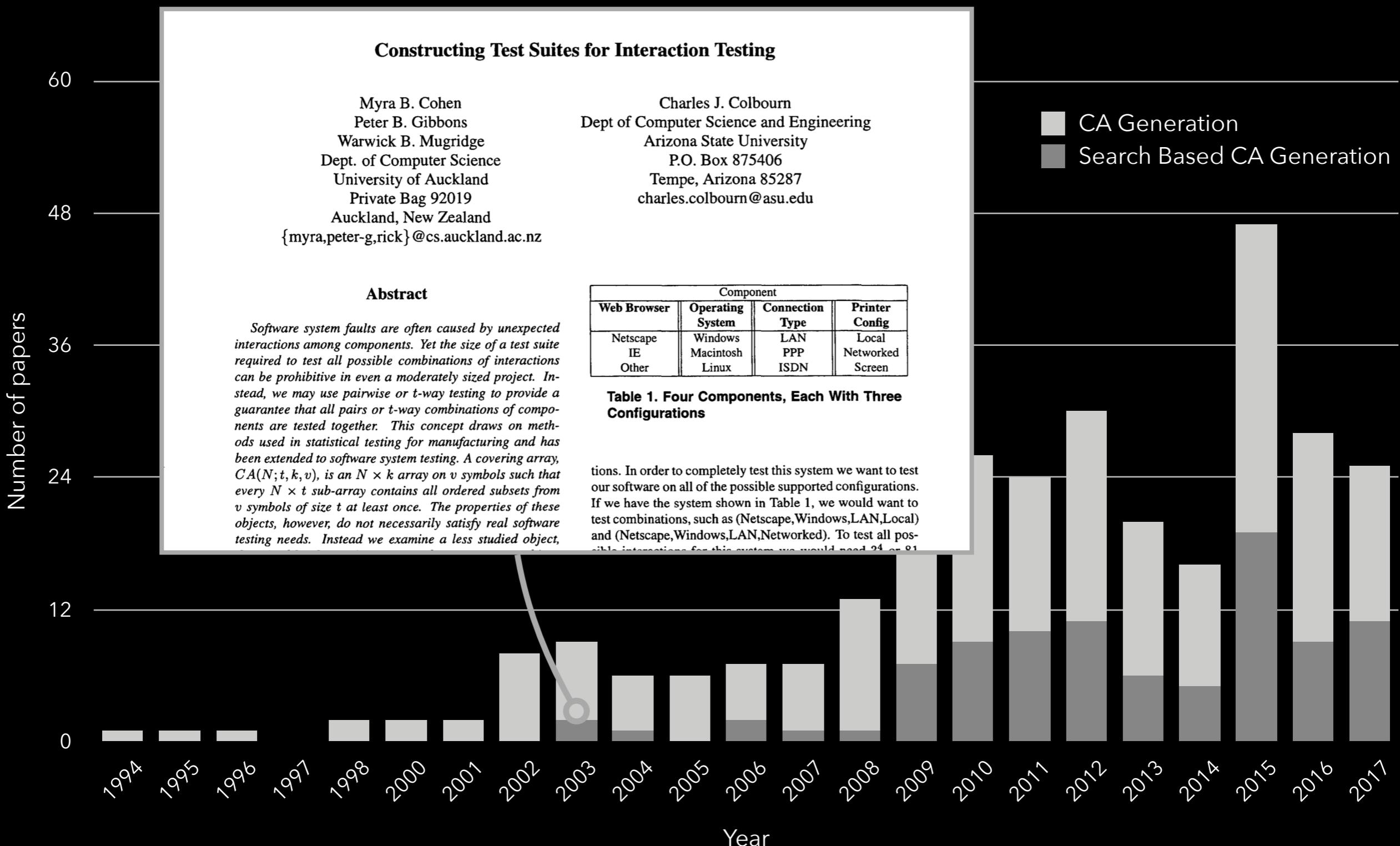
Tabu Search

Genetic Algorithm

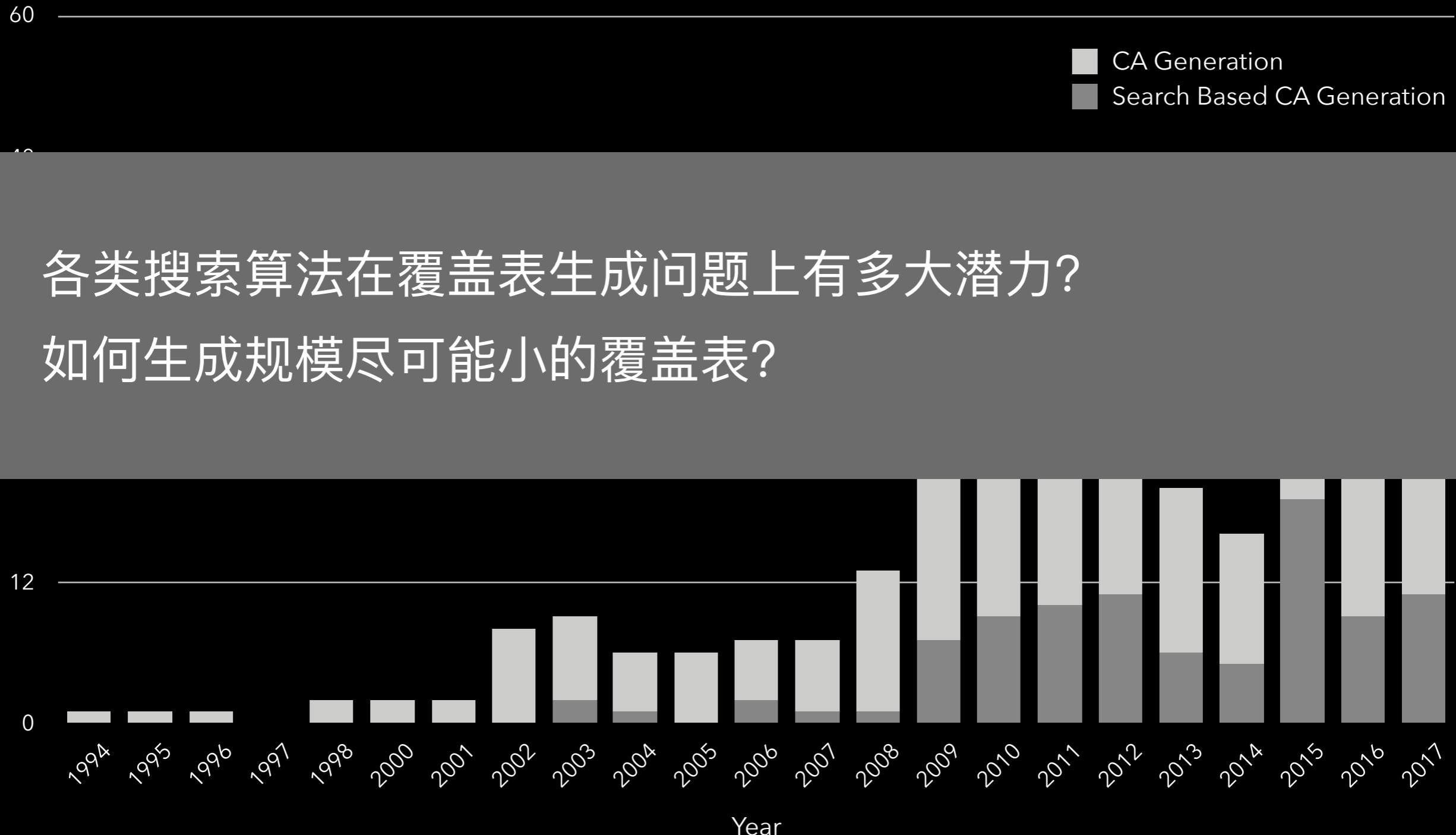
Particle Swarm Optimisation

Ant Colony Optimisation

基于搜索的覆盖表生成



基于搜索的覆盖表生成



基于搜索的覆盖表生成

One-Test-at-a-Time + GA

参数调优 通过实验设计方法确定最适合于覆盖表生成的算法
参数和搜索算子配置

种群规模	进化代数	交叉概率	变异概率
100	100	1	1
2100	600	0.8	0.8
4100	1100	0.6	0.6
6100		0.4	0.4
		0.2	0.2

测试号	a_1	a_2	a_3	a_4	a_5	测试号	a_1	a_2	a_3	a_4	a_5
B_1	5	0	2	1	4	B_{11}	5	0	1	1	4
B_2	0	0	2	1	4	B_{12}	5	0	2	0	4
B_3	1	0	2	1	4	B_{13}	5	0	2	2	4
B_4	2	0	2	1	4	B_{14}	5	0	2	3	4
B_5	3	0	2	1	4	B_{15}	5	0	2	4	4
B_6	4	0	2	1	4	B_{16}	5	0	2	1	0
B_7	5	1	2	1	4	B_{17}	5	0	2	1	1
B_8	5	2	2	1	4	B_{18}	5	0	2	1	2
B_9	5	3	2	1	4	B_{19}	5	0	2	1	3
B_{10}	5	0	0	1	4						

基于搜索的覆盖表生成

One-Test-at-a-Time + ACO

参数调优 通过实验设计方法确定最适合于覆盖表生成的算法
参数和搜索算子配置

		AS	ACS	MMAS
信息素指数 a	$0 < a \leq 5$	√	√	√
信息素指数 b	$0 < b \leq 5$	√	√	√
信息素残留指数 p	$0 < p \leq 1$	√	√	√
蚂蚁数量 m	> 0	√	√	√
迭代次数 Nc	> 0	√	√	√
信息素初始值 init	$0 \leq \text{init} \leq 10$	√	√	√
路径选择概率 q0	$0 < q0 < 1$		√	
局部更新系数 ql	$0 < ql < 1$		√	
信息素上界 max	$\min < \text{max} < 5$			√
信息素下界 min	$0 < \text{min} < \text{max}$			√

表 16 AETG-MMAS 二维实验结果											
	MCA1	CA2	CA3	MCA4	MCA5	CA6	CA7	CA8	MCA9	MCA10	总和
C1	97	106	68	96	94	31	41	18	48	113	712
C2	95	102	63	92	94	30	42	18	47	112	695
C3	95	103	66	95	95	31	42	18	48	113	706
C4	95	103	66	94	93	32	42	18	48	111	702
C5	94	103	63	93	93	30	42	17	47	114	696
C6	92	94	61	86	91	30	42	18	47	112	673
C7	94	100	63	89	92	31	41	18	49	114	691
C8	95	99	63	92	92	31	42	18	47	111	690
C9	95	103	65	94	95	31	41	18	49	112	703
C10	94	106	68	95	95	31	42	17	48	113	709
C11	95	96	62	88	92	31	42	18	48	113	685
C12	94	95	62	87	91	31	43	17	48	114	682
C13	94	98	62	86	92	31	41	17	48	114	683
C14	93	98	63	90	92	31	42	18	47	112	686
C15	94	104	65	94	94	31	43	18	46	113	702
C16	96	104	67	96	94	30	42	17	49	113	708
C17	96	103	64	93	96	31	41	18	49	113	704
C18	96	103	66	95	95	31	42	18	48	111	705
C19	95	104	65	94	95	31	42	17	49	113	705
C20	97	103	64	94	93	31	42	19	47	114	704
C21	94	98	62	89	93	30	42	18	47	111	684
C22	95	105	67	95	94	31	41	18	48	113	707
C23	94	93	62	86	93	30	39	17	48	113	675
C24	97	103	66	94	95	31	41	18	48	114	707

基于搜索的覆盖表生成

One-Test-at-a-Time + GA / ACO / PSO

传统方法 直接应用搜索算法来解决覆盖表生成问题

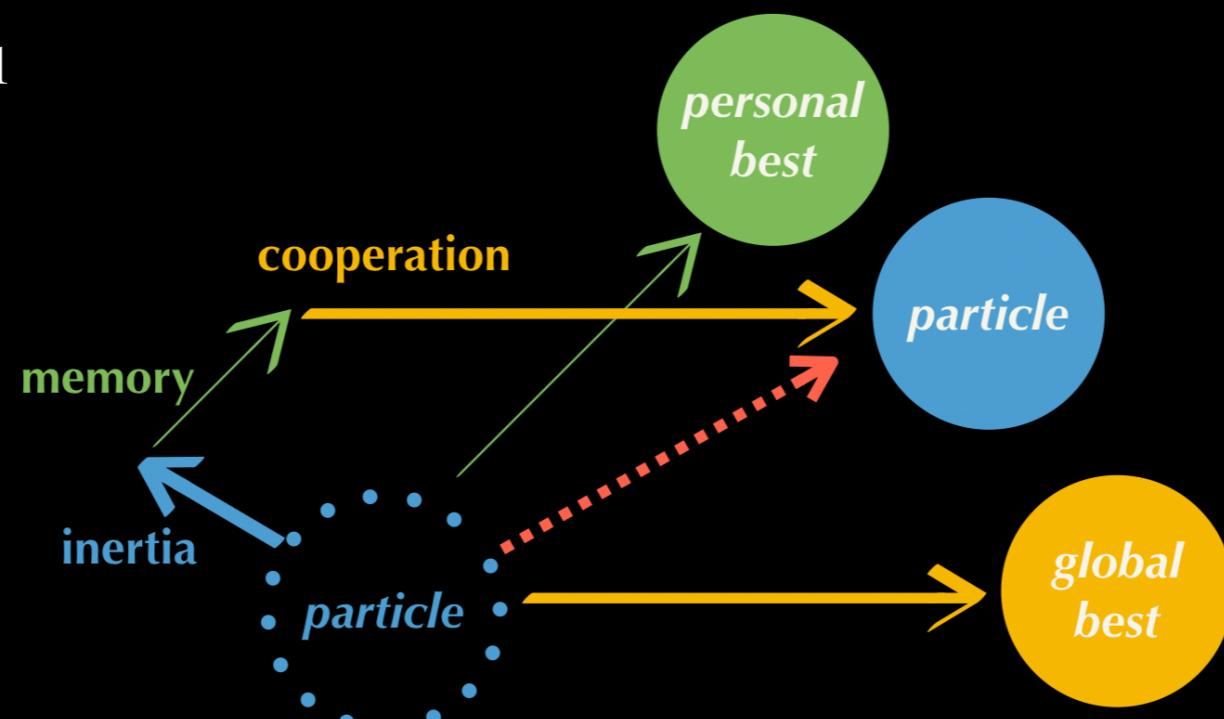
基于搜索的覆盖表生成

One-Test-at-a-Time + PSO

传统方法 直接应用粒子群算法来逐条生成测试用例

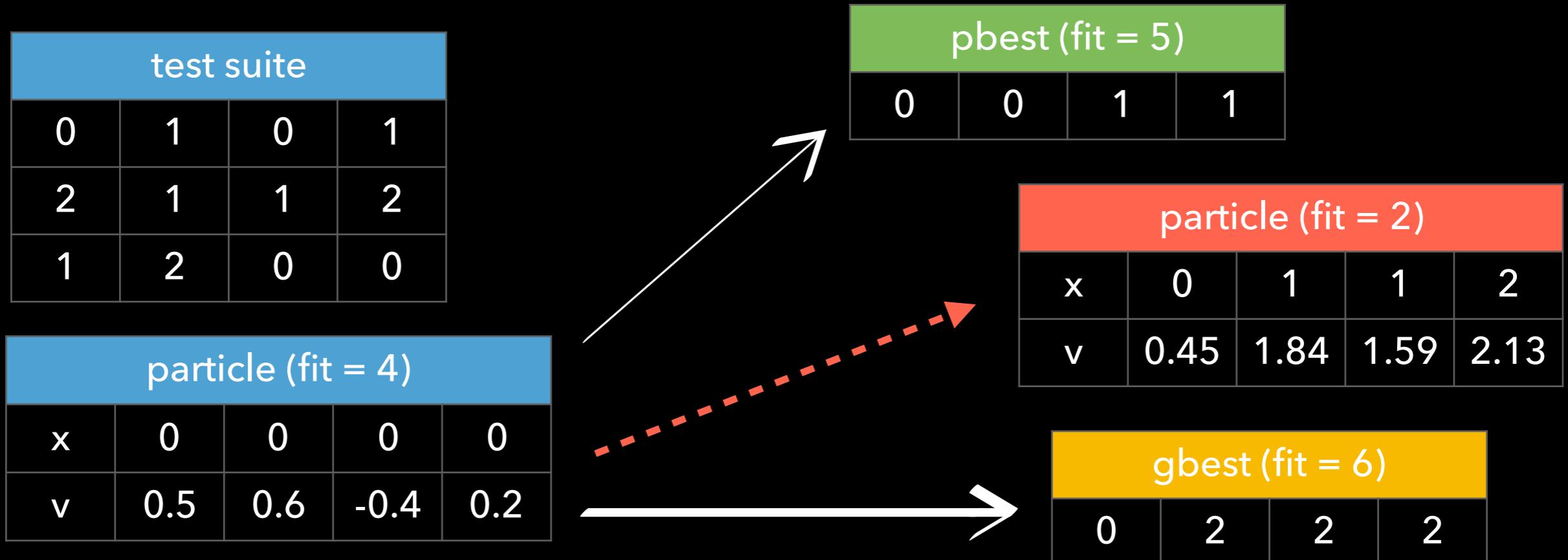
$$v_{i+1} = \omega \times v_i + c_1 \times r_1 \times (pbest_i - x_i) + c_2 \times r_2 \times (gbest - x_i)$$

$$x_{i+1} = x_i + v_{i+1}$$



基于搜索的覆盖表生成

传统粒子群算法中每个粒子都将向当前最优位置的方向移动，但距离最优位置“更近”的粒子并不一定能覆盖更多组合



基于搜索的覆盖表生成

传统粒子群算法中每个粒子都将向当前最优位置的方向移动，但距离最优位置“更近”的粒子并不一定能覆盖更多组合

离散粒子群算法

- Swap operator-based PSO
- Space transformation-based PSO
- Fuzzy matrix-based PSO
- Probability-based PSO (S-PSO)
- Incorporating-based PSO

基于搜索的覆盖表生成

离散粒子群算法 DPSO 为覆盖表生成这一离散空间中的优化问题设计更为合适的演化算子

particle				
position	0	0	0	0
velocity	(-, -, 2, 2)		0.7	a possible t -way combination
	(0, -, 0, -)		0.5	
	(-, 0, -, 1)		0.8	
	

the probability of selecting this combination to update the position

基于搜索的覆盖表生成

离散粒子群算法 DPSO 为覆盖表生成这一离散空间中的优化问题设计更为合适的演化算子

以覆盖更多组合为目标设计粒子演化算子

particle					
x	0	0	0	0	
v	(-, -, 2, 2)			0.7	
	(0, -, 0, -)			0.5	
	(-, 0, -, 1)			0.8	
	

gbest			
0	2	2	2



(I) update v
learn uncovered combinations from
lbest and gbest, e.g., $(-, -, 2, 2)$.

基于搜索的覆盖表生成

离散粒子群算法 DPSO 为覆盖表生成这一离散空间中的优化问题设计更为合适的演化算子

以覆盖更多组合为目标设计粒子演化算子

particle					
x	0	0	2	2	
v	(-, -, 2, 2)			0.7	
	(0, -, 0, -)		0.5		
	(-, 0, -, 1)		0.8		
		

∴
∴

gbest			
0	2	2	2

(2) update x
make x cover new combinations
with a certain probability

基于搜索的覆盖表生成

在 88 个统一强度和可变强度覆盖表上评估 DPSO 的性能

- 同已有研究中最优的粒子群算法比较 (CPSO)
- 同已有研究中其它搜索算法比较 (GA & ACO)

74%

DPSO 在覆盖表生成规模的平均值上显著优于 CPSO 算法

100%

DPSO 能生成比 GA 和 ACO 规模更小的覆盖表

基于搜索的组合测试

使用高效的启发式搜索技术来解决组合测试中的各类问题

1. 组合测试用例生成问题
2. 组合测试用例优先级排序问题
3. 组合测试有效性问题

基于搜索的测试用例集排序

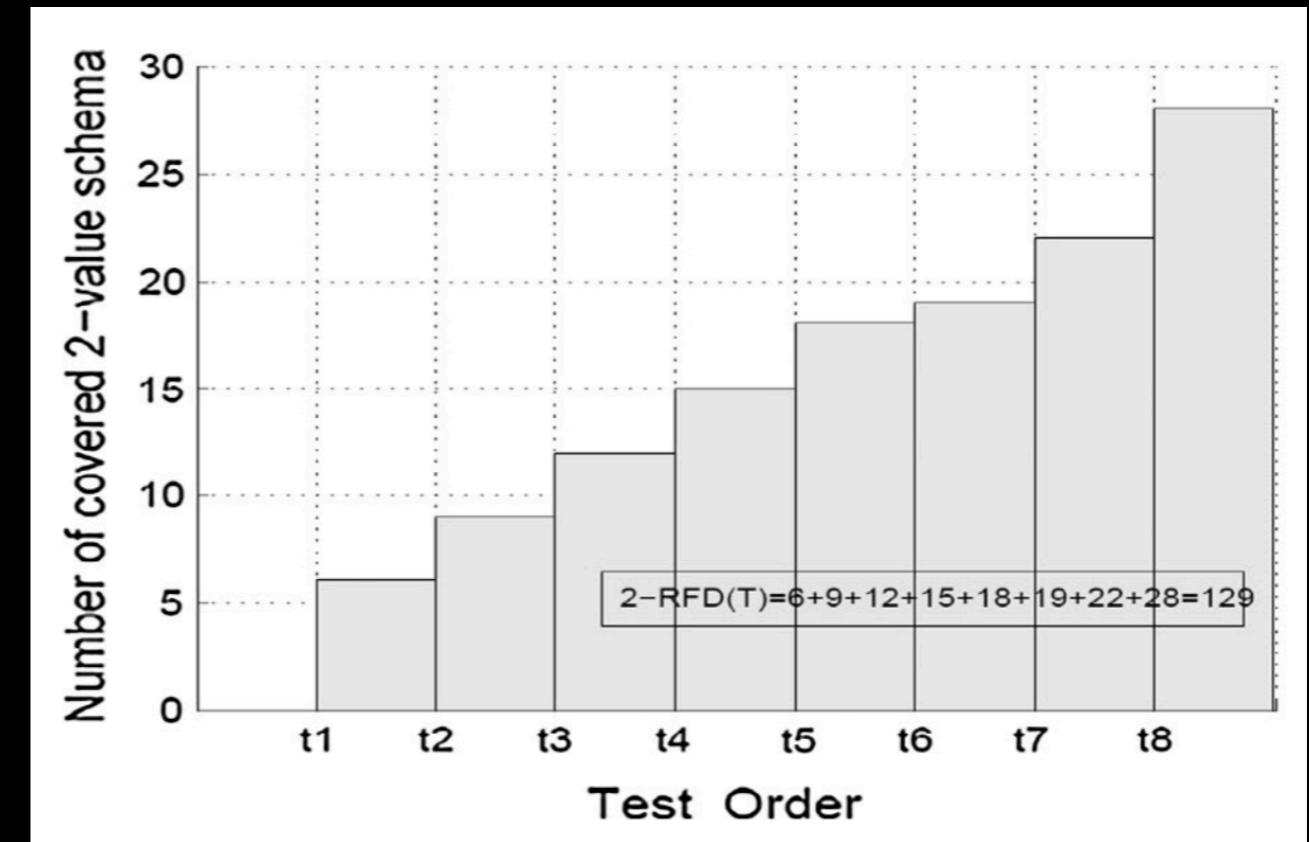
测试用例优先级排序

- 寻找故障检测速率 (APFD) 最高的测试执行序列
- 通常根据覆盖准则来设计排序目标：组合覆盖

t_1	0	1	1	0
t_2	0	1	0	1
t_3	1	1	0	0
t_4	1	0	1	0
t_5	0	0	0	1

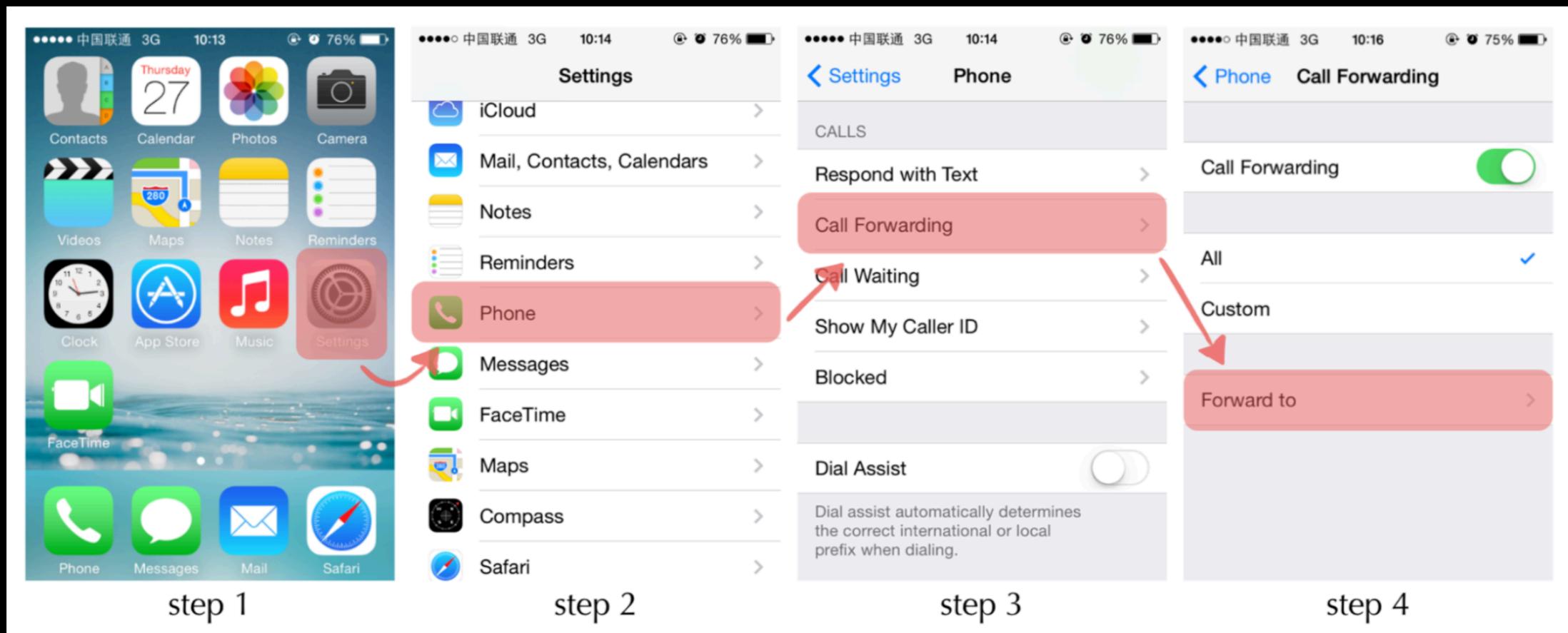


$t_3 \rightarrow t_2 \rightarrow t_5 \rightarrow t_4 \rightarrow t_1$



基于搜索的测试用例集排序

切换成本 (switching cost) 在有些待测软件中，更改相邻测试用例间参数的取值可能会导致重配置开销



基于搜索的测试用例集排序

切换成本 (switching cost) 在有些待测软件中，更改相邻测试用例间参数的取值可能会导致重配置开销

	P_1	P_2	P_3	P_4
weight	1	5	2	5

$$d_{i,j} = \sum_{k=1}^n \omega_k \times \theta_k$$

t_1	0	0	1	1
t_2	0	1	2	1

从 t_1 切换到 t_2 的成本为 $5 + 2 = 7$

基于搜索的测试用例集排序

最小化切换成本 → 下一条测试用例尽可能相同

最大化组合覆盖 → 下一条测试用例尽可能不同

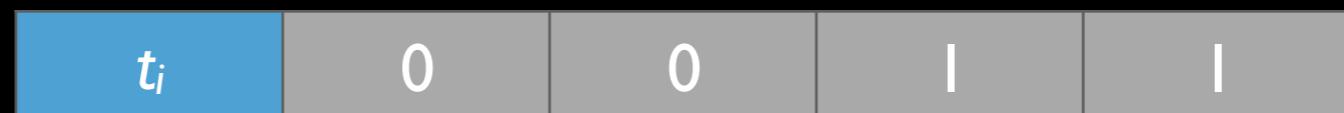


t_a	1	2	0	2
t_b	2	1	0	1
t_c	0	2	1	1
t_d	1	1	0	0
t_e	0	0	0	0

基于搜索的测试用例集排序

最小化切换成本 \rightarrow 下一条测试用例尽可能相同

最大化组合覆盖 \rightarrow 下一条测试用例尽可能不同



t_a	1	2	0	2
t_b	2	1	0	1
t_c	0	2	1	1
t_d	1	1	0	0
t_e	0	0	0	0

基于搜索的测试用例集排序

基于组合覆盖的排序 (maximise combination coverage)

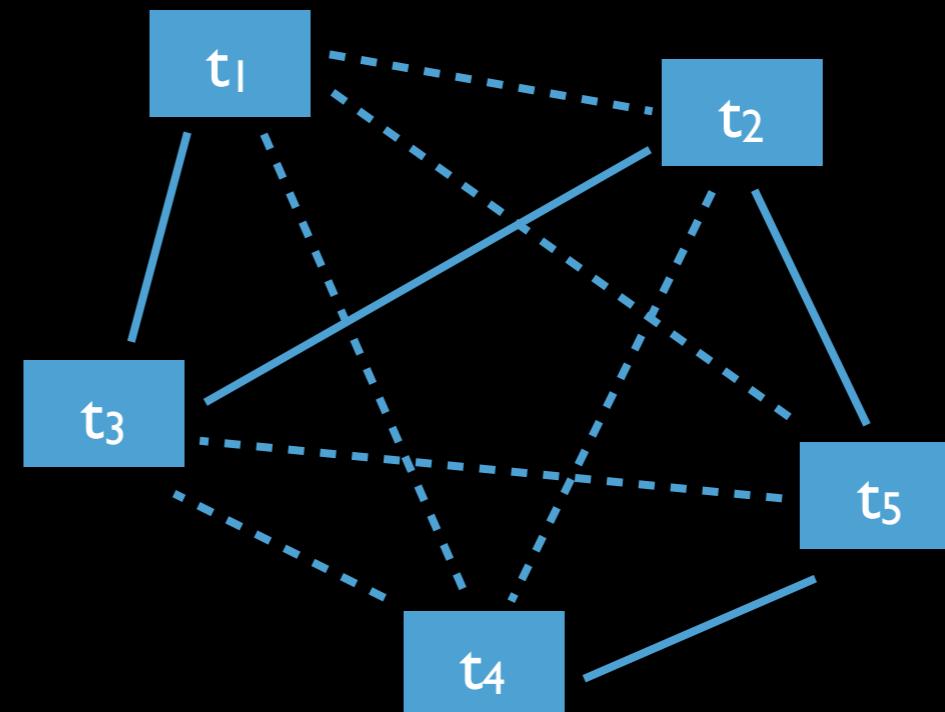
基于搜索的测试用例集排序

基于组合覆盖的排序 (maximise combination coverage)

基于切换成本的排序 (minimise switching cost)

降低切换成本有助于尽早执行完所有测试用例: Greedy / GA / TSP Solver

t_1	1 0 0 0 1
t_2	1 1 1 1 0
t_3	0 1 0 0 1
t_4	0 0 1 1 1
t_5	1 1 0 1 1



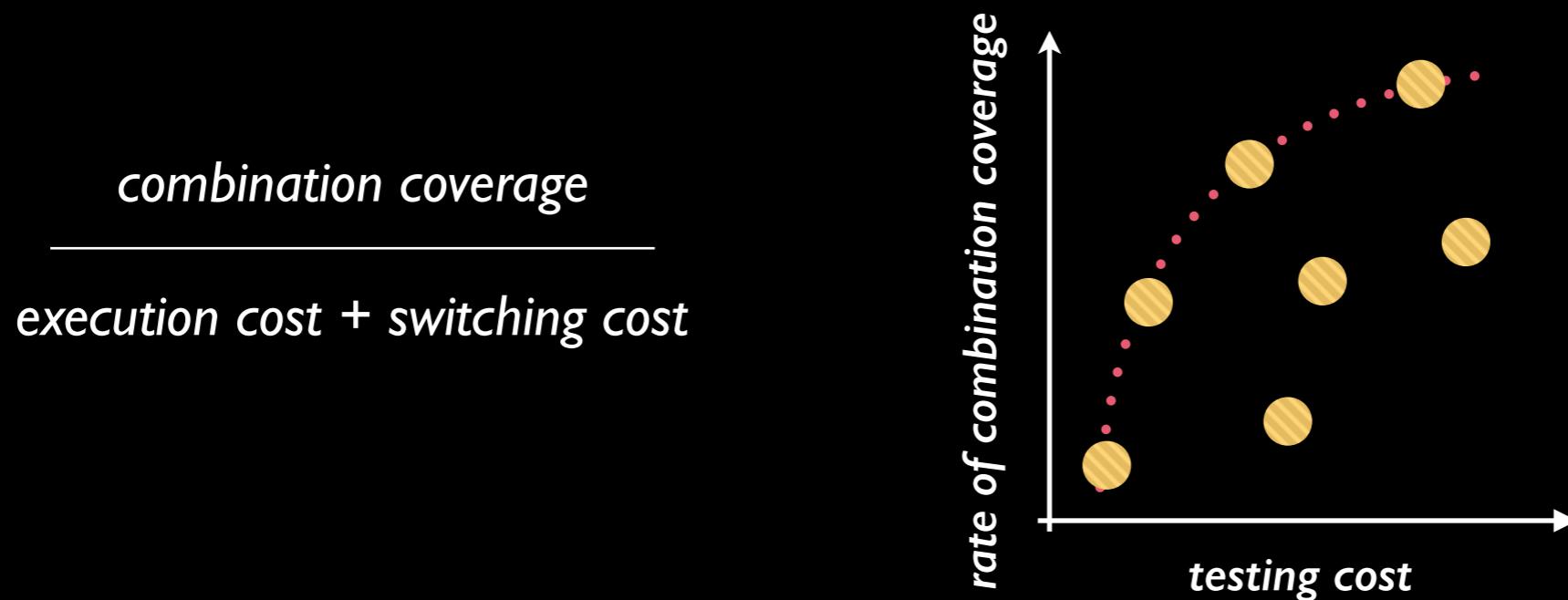
基于搜索的测试用例集排序

基于组合覆盖的排序 (maximise combination coverage)

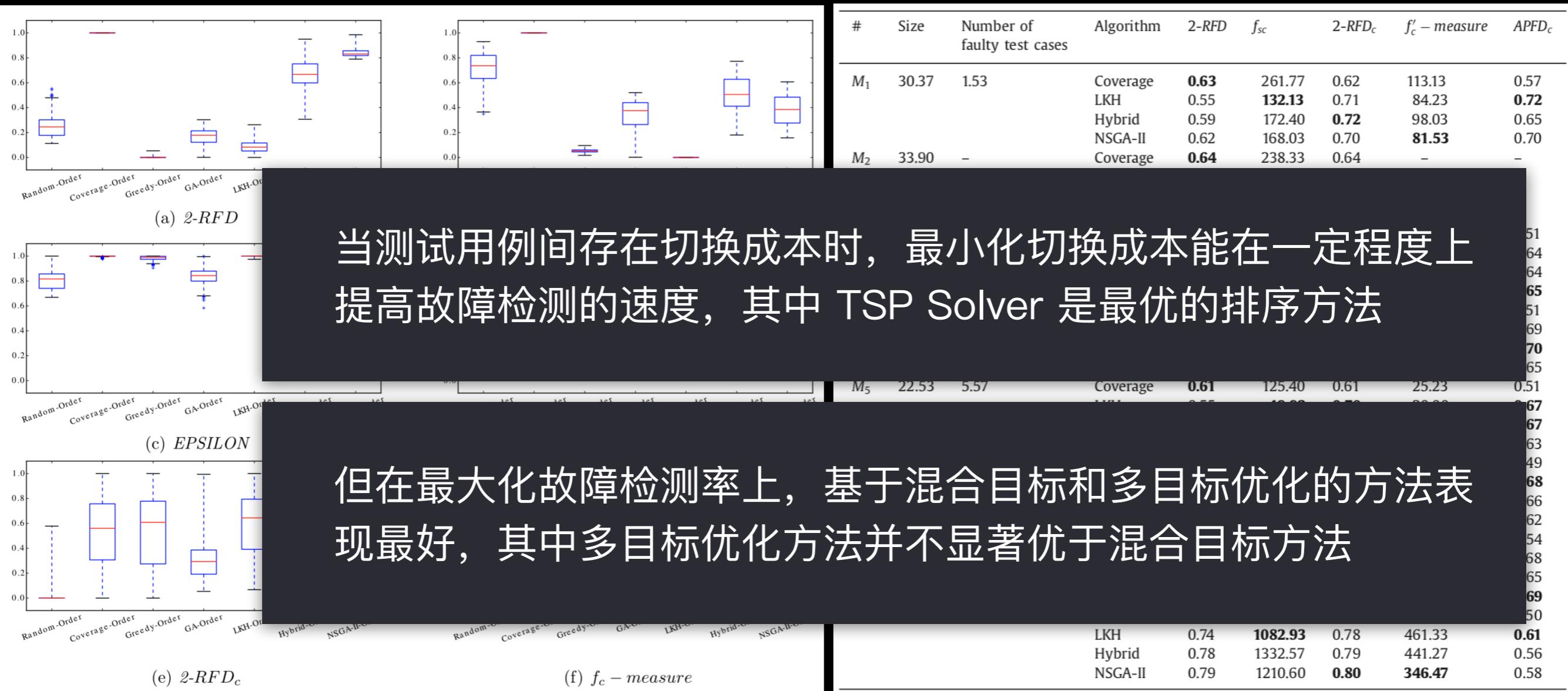
基于切换成本的排序 (minimise switching cost)

基于混合目标和多目标优化的排序 (balance coverage & cost)

平衡组合覆盖和测试成本: Hybrid / NSGA-II



基于搜索的测试用例集排序



当测试用例间存在切换成本时，最小化切换成本能在一定程度上提高故障检测的速度，其中 TSP Solver 是最优的排序方法

但在最大化故障检测率上，基于混合目标和多目标优化的方法表现最好，其中多目标优化方法并不显著优于混合目标方法

基于搜索的组合测试

使用高效的启发式搜索技术来解决组合测试中的各类问题

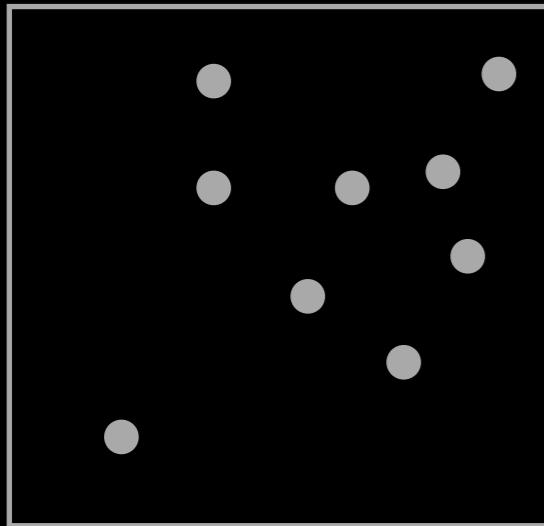
1. 组合测试用例生成问题
2. 组合测试用例优先级排序问题
3. 组合测试有效性问题

组合测试的有效性

基于搜索的测试用例生成：从所有潜在可能的测试用例 (search space) 中找出最满足某个测试目标 (fitness) 的方案

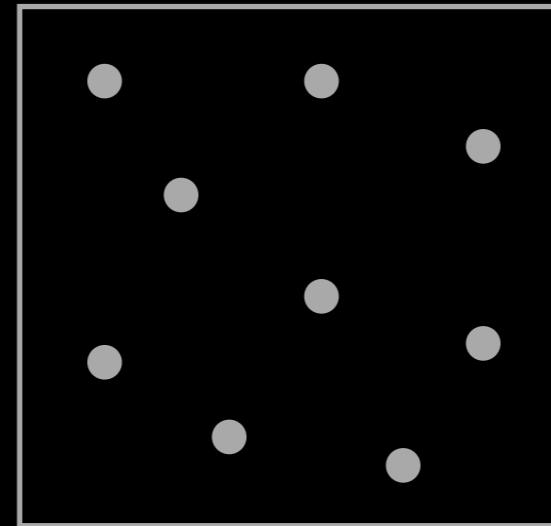
Random Testing

随机选择测试用例



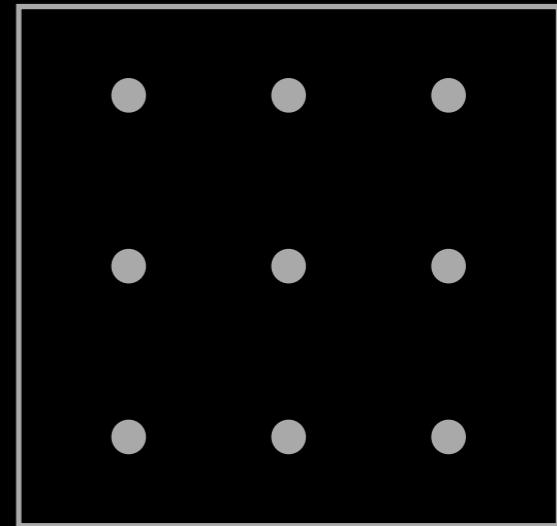
Adaptive Random Testing

最大化测试用例集的多样性
(maximise diversity)



Combinatorial Testing

最大化测试用例集的组合
覆盖率 (maximise
combination coverage)



组合测试的有效性

组合测试是否能检测到比 RT 和 ART 更多的软件故障？

CT 明显的优于 RT

[Dunietz 97] [Kobayashi 01] [Bell 05] [Pretschner 08]
[Ballance 12] [Vilkomir 13] [Calvagna 15] [Medeiros 16]

已有工作均假设测试人员能为待测软件建立完整的测试模型，这可能潜在
夸大了特定测试方法在实践中的故障检测能力

具有不同检测难度的故障会为测试方法带来不同的挑战

在相同的测试用例集规模下，
ART 也能覆盖超过 90% 的组合

[Henard 14]

组合测试的有效性

组合测试在什么情况下能检测到比 RT 和 ART 更多的软件故障？

测试场景

一个测试模型和对应的一组具有相同检测难度的故障

<para, cons, rate>

模型中完整识别的参数比例 $\text{pare} = \{0.4, 0.6, 0.8, 1.0\}$

模型中完整识别的约束比例 $\text{cons} = \{0.4, 0.6, 0.8, 1.0\}$

待测故障的故障率 $\text{rate} = \# \text{ failure inputs} / \# \text{ all inputs}$

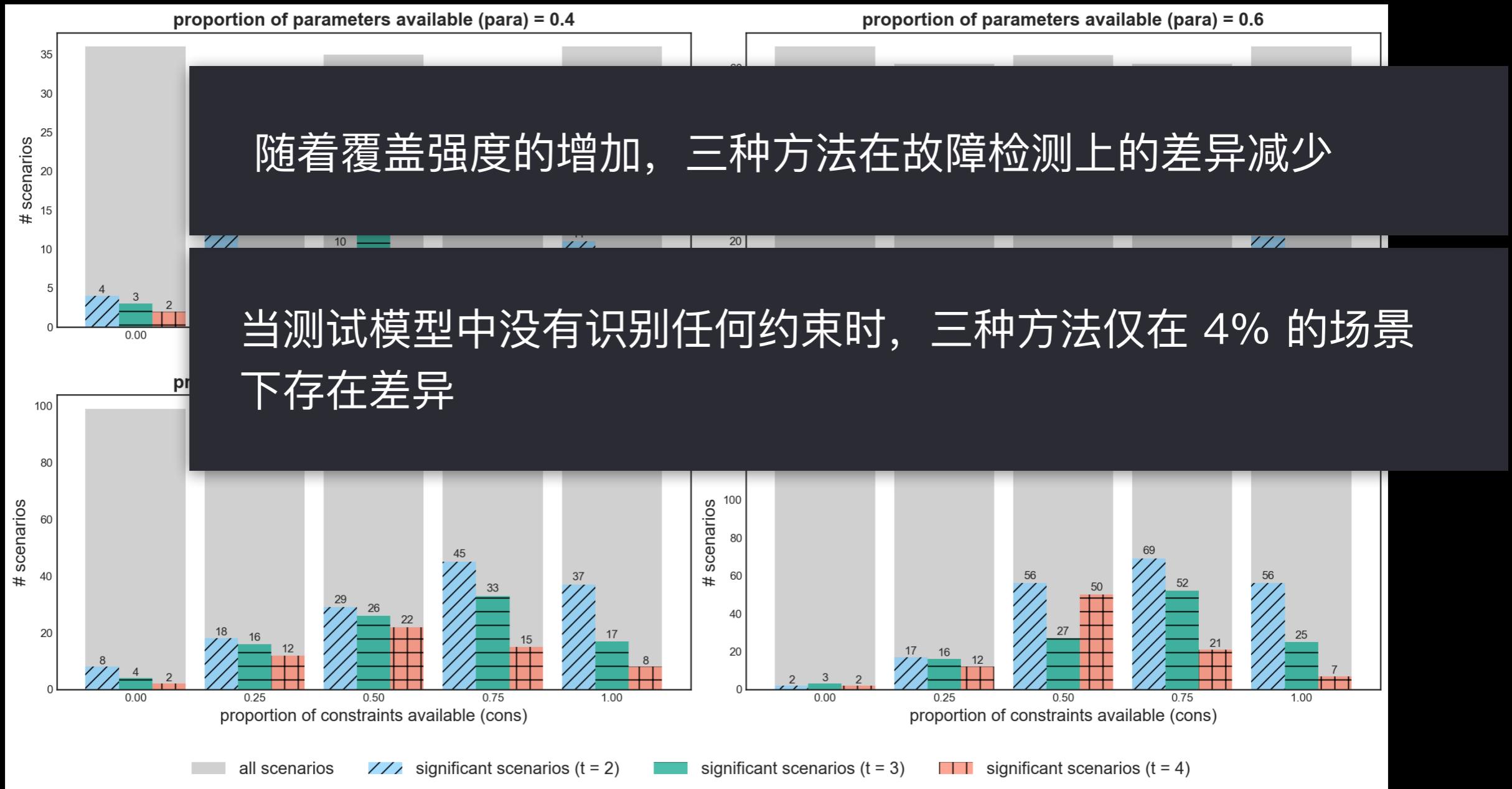
组合测试的有效性

九个真实软件系统，完整测试模型和故障语料均来自已有工作
[Petke 2015] [Medeiros 2016] [Sanchez 2017]

总计定义 1683 个测试场景

Name	Description	LOC	Model	# Parameters (n)	# Constraints	# Faults	# Scenarios
FLEX	lexical analyser	15,297	$2^2 3^2 2^4 5^1$	9	12	50	270
GREP	text-search utility	15,633	$3^2 4^1 6^1 8^1 4^1 3^1 2^1 5^1$	9	83	12	125
GZIP	compression utility	6,582	$2^{13} 3^1$	14	61	5	70
SED	stream text editor	11,148	$2^4 6^1 10^1 2^1 4^1 2^2 3^1$	11	50	22	250
MAKE	build utility	27,879	2^{10}	10	1	2	14
NANOXML	XML parser	7,646	$2^5 4^1 2^1$	7	6	16	54
DRUPAL	web framework	336,025	2^{47}	47	45	160	530
BUSYBOX	UNIX utilities	189,722	2^{68}	68	16	9	120
LINUX	operation system	12,594,584	2^{104}	104	83	28	250

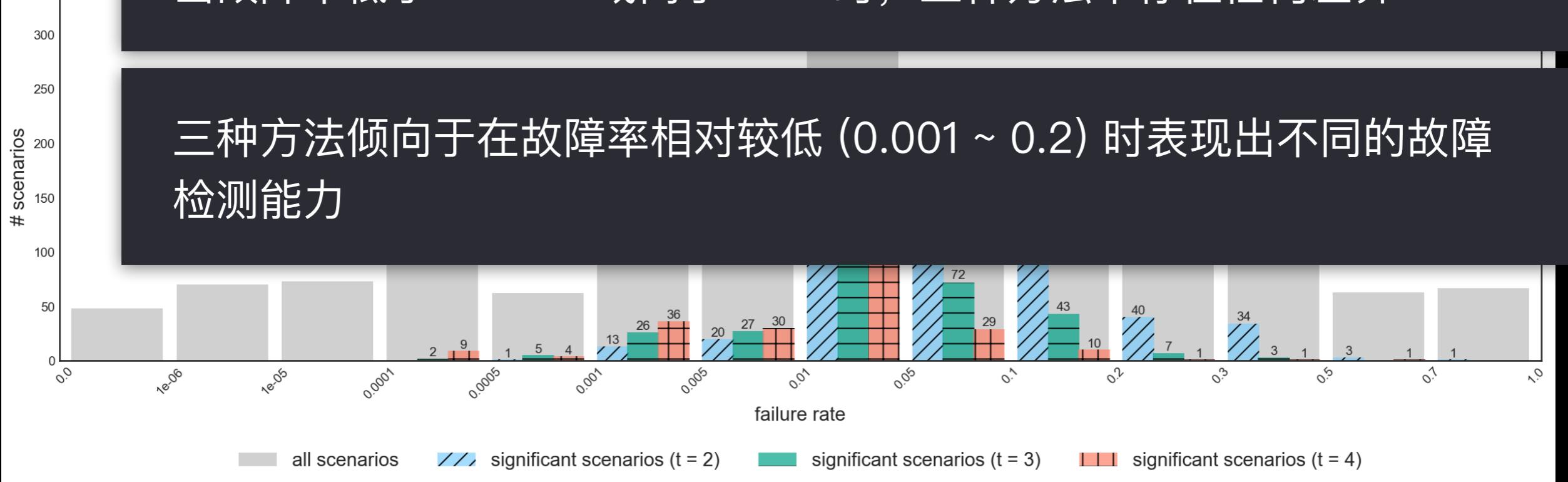
组合测试的有效性



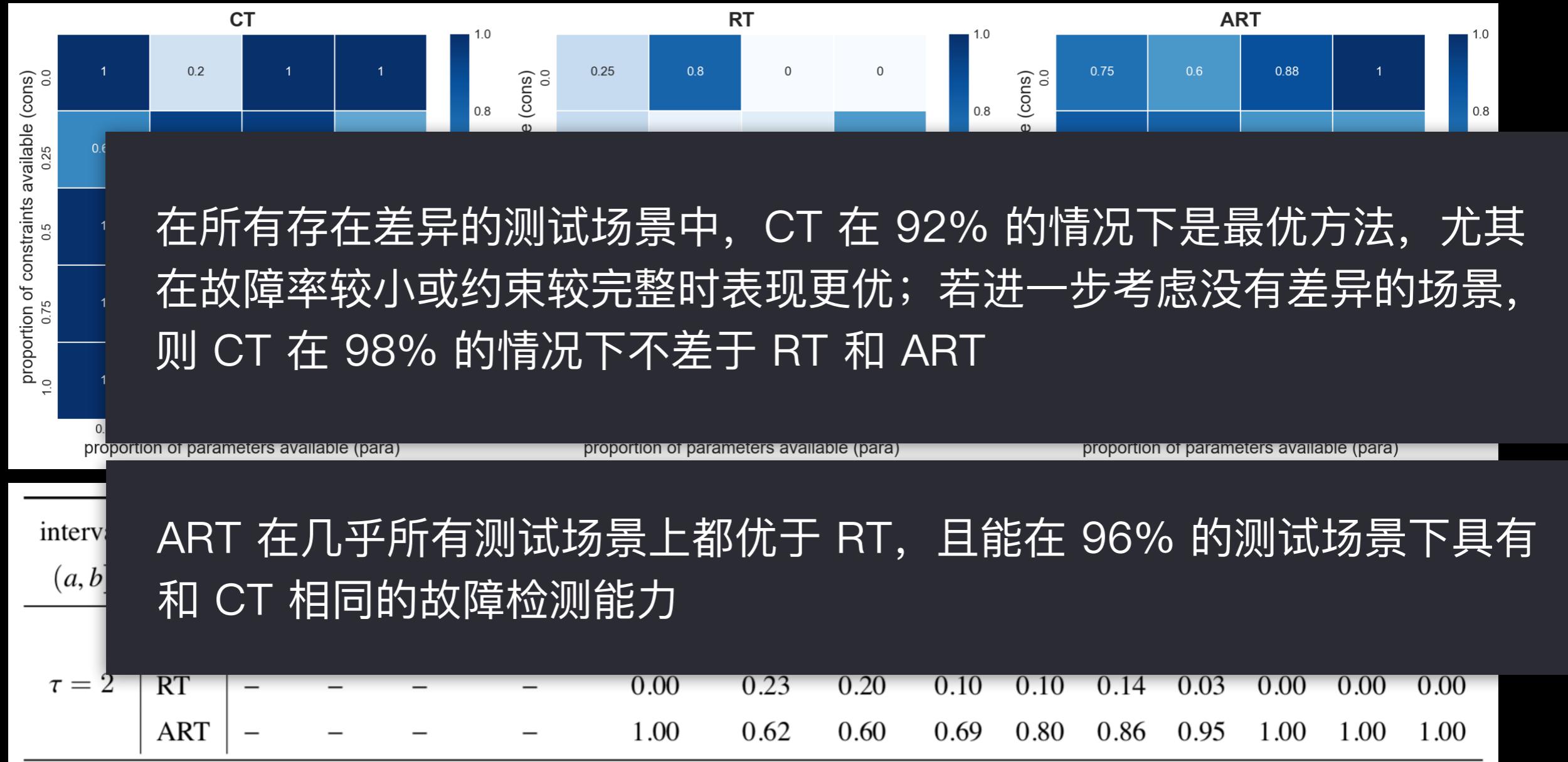
组合测试的有效性

当故障率低于 0.0001 或高于 0.71 时，三种方法不存在任何差异

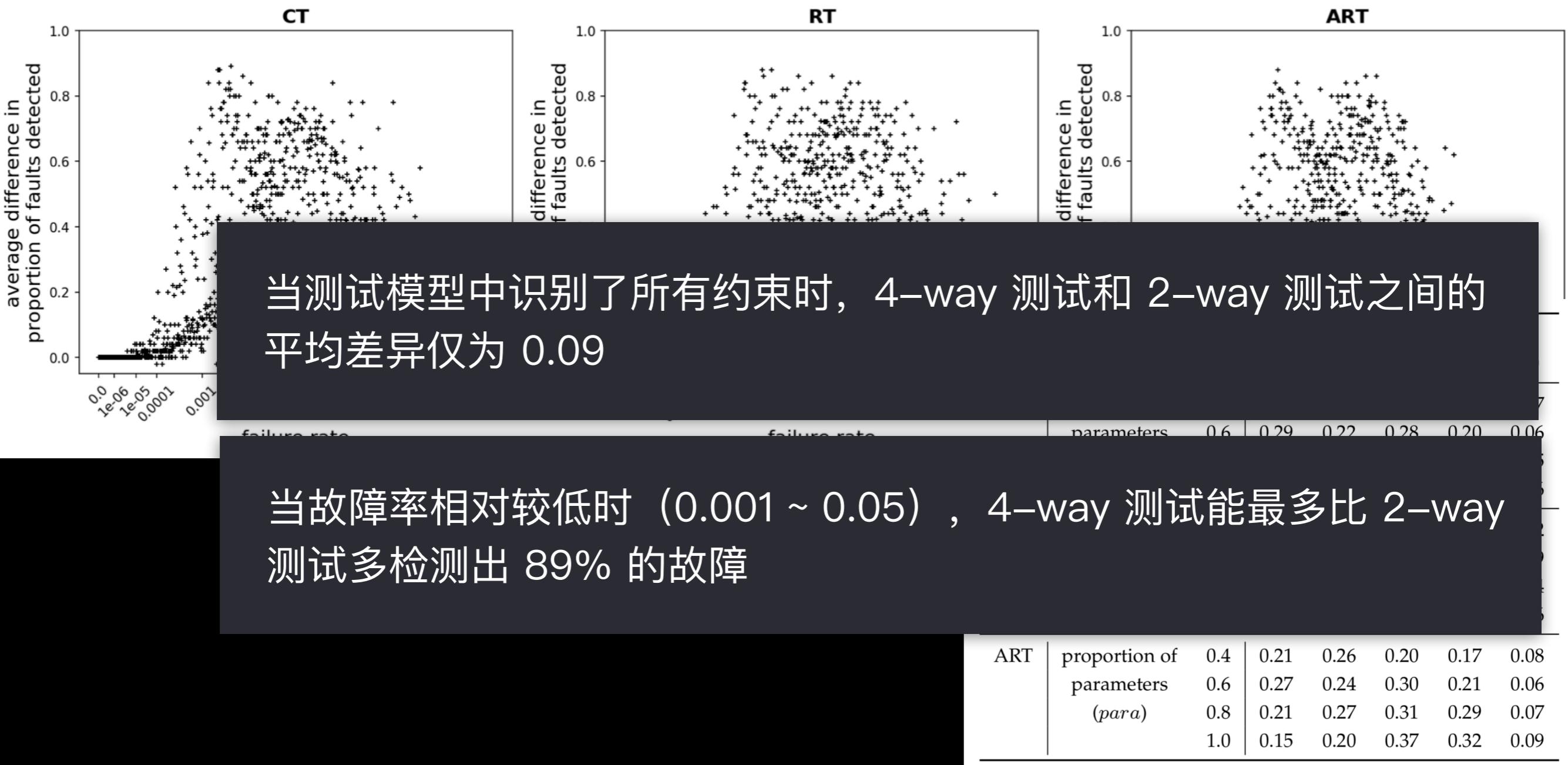
三种方法倾向于在故障率相对较低 (0.001 ~ 0.2) 时表现出不同的故障检测能力



组合测试的有效性



组合测试的有效性



基于搜索的组合测试

使用高效的启发式搜索技术来解决组合测试中的各类问题

1. 组合测试用例生成问题
2. 组合测试用例优先级排序问题
3. 组合测试有效性问题

SBCT 的未来研究 (open problems and challenges)

基于搜索的组合测试

I. Constrained Covering Array

约束广泛存在于现实世界的待测系统中，约束覆盖表因而是一种更实用的测试用例集

- 覆盖任意 t 个参数间的所有合法组合至少一次
- 任意一条测试用例都不包含违反约束的组合

strikeththrough	double strikeththrough	super script	sub script	small caps	all caps	hidden
0	0	0	0	0	0	0
1	1	1	1	1	1	1

Constraint: *superscript* and *subscript* cannot both be enabled.

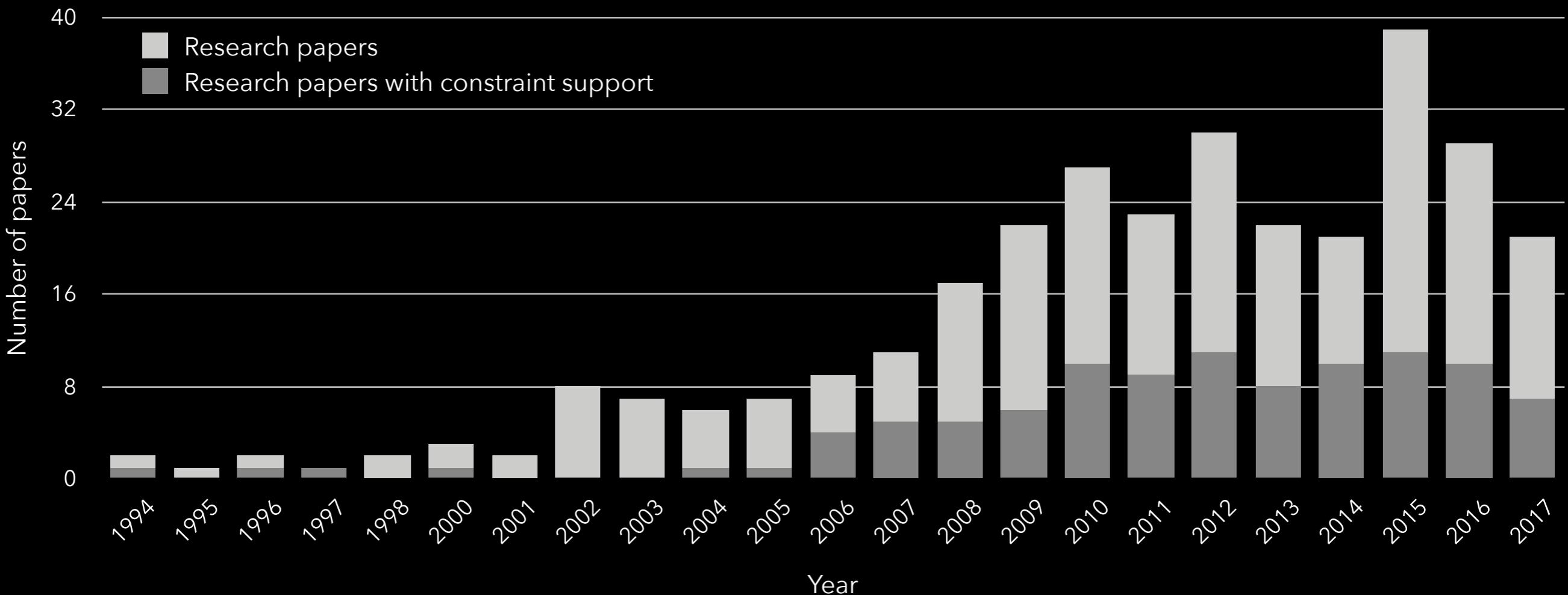
strikeththrough	double strikeththrough	super script	sub script	small caps	all caps	hidden
0	1	1	0	1	1	0
1	0	0	0	0	0	1
1	1	0	1	1	0	0
0	0	0	1	0	1	1
1	1	1	0	0	1	1
0	0	1	0	0	0	0
0	0	0	1	1	1	1

基于搜索的组合测试

I. Constrained Covering Array

在所有与覆盖表生成相关的研究中，仅有 33% 考虑了对约束的处理

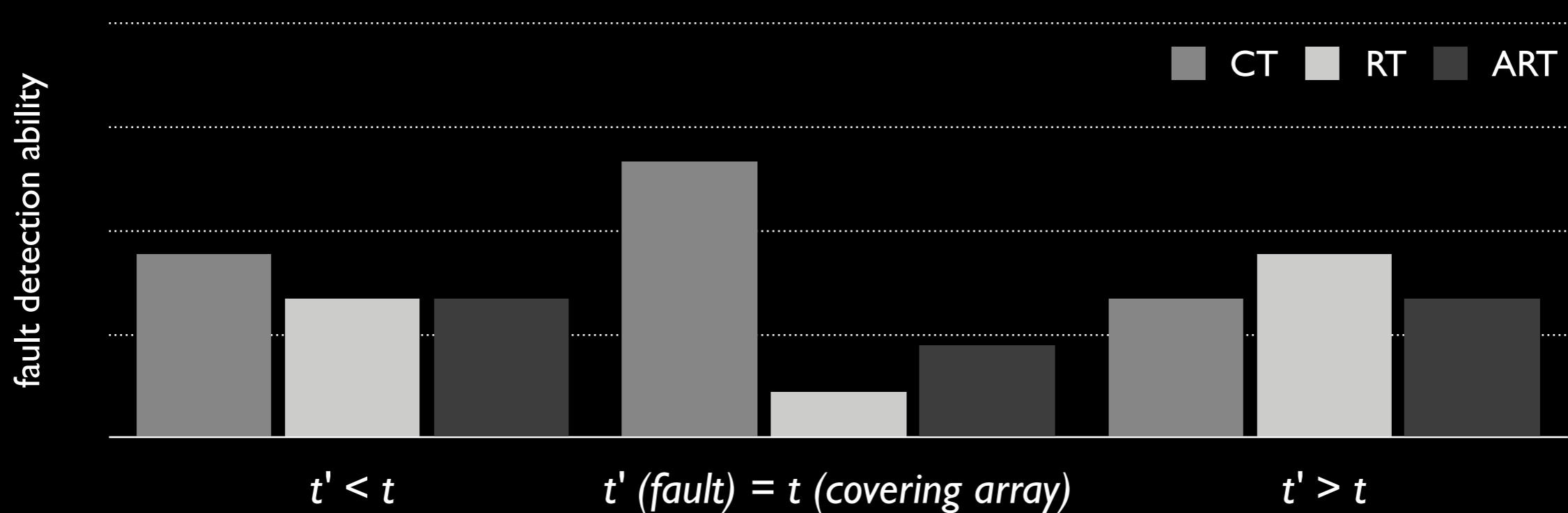
尽管已有很多约束处理方法，但如何更高效地处理约束仍有待研究



基于搜索的组合测试

2. New Objectives of Covering Array Generation

使用 t -way 覆盖表能有效检测由不超过 t 个参数间相互作用所引发的故障，但当与故障相关的参数个数超过 t 时，仅使用 t -way 覆盖表则往往不如人们预想的有效



基于搜索的组合测试

2. New Objectives of Covering Array Generation

最小化 t -way covering array 的规模并不总意味着最大化故障检测能力，如何设计更加合适的优化目标 (fitness function) 仍有待研究

- 适当增加组合覆盖的多样性
- t -way fault profile [Kuhn 2017]

基于搜索的组合测试

3. New CT Problems

组合测试建模问题：确定影响系统行为的参数及其之间的关系

- 模型抽取和建立
- 模型维护

准确和完整的建模是组合测试有效性的前提，但仍缺乏有效的自动化或半自动化技术和辅助工具

- Collective Intelligence (Machine + Human)

Thanks

吴化尧

南京大学

hhywu@nju.edu.cn

CSBSE 2018