南京大學
NANJING UNIVERSITY

Enhance Combinatorial Testing with Metamorphic Relations

xintao niu
niuxintao@nju.edu.cn

Combinatorial Testing

- Modern Software is complex, configurable, interactive
- Testing Such System is challenging when considering the large testing space
- A reasonable requirement is to construct an elaborate test suite with small size.

Combinatorial Testing

- A simple Example is a table

p_1	p_2	p_3
0	0	0
0	1	1
1	0	1
1	1	0

Combinatorial Testing

- A simple Example is a table

p_1	p_2	p_3
0	0	0
0	1	1
1	0	1
1	1	0

Combinatorial Testing

- A simple Example is a table

p_1	p_2	p_3
0	0	0
0	1	1
1	0	1
1	1	0

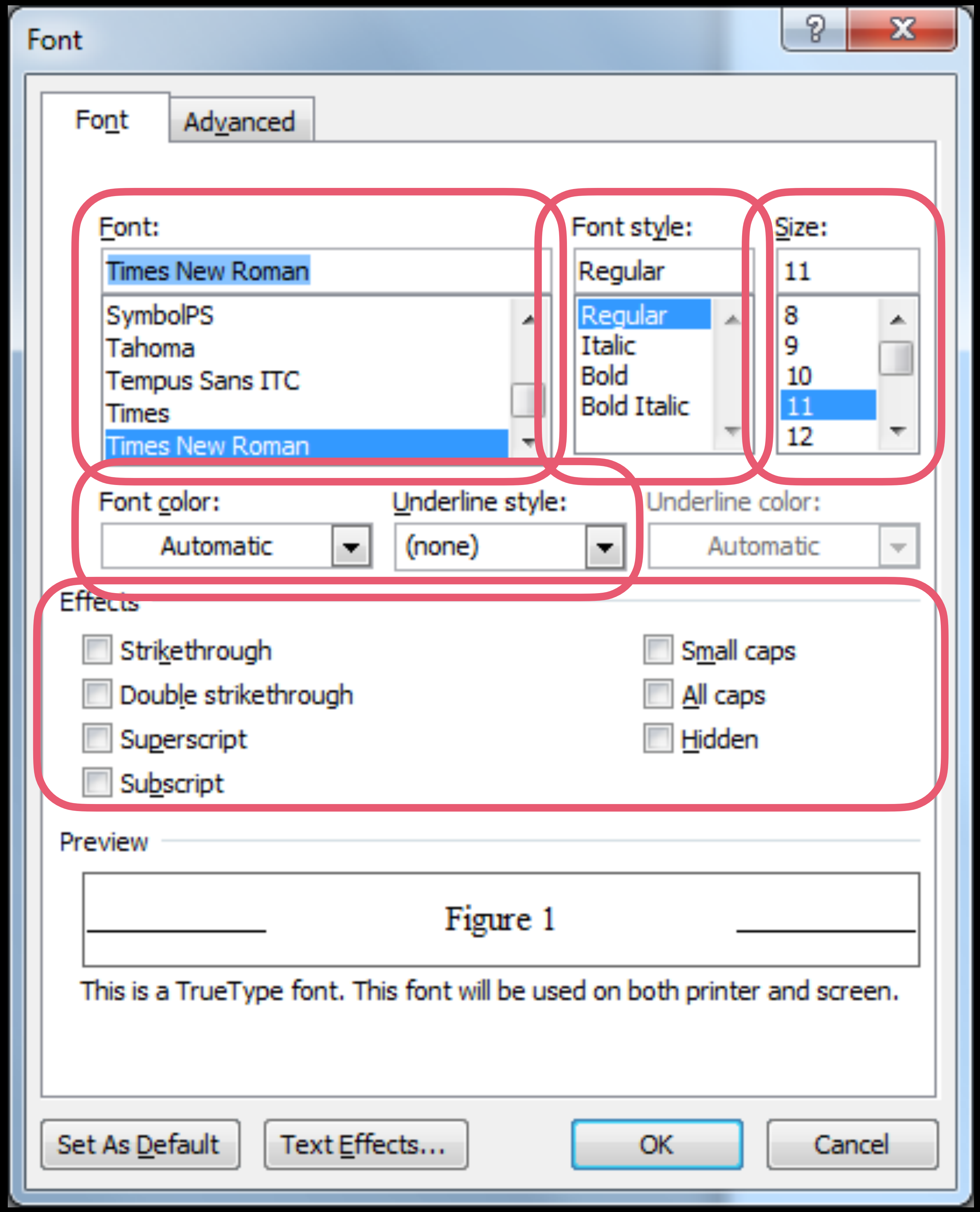
Combinatorial Testing

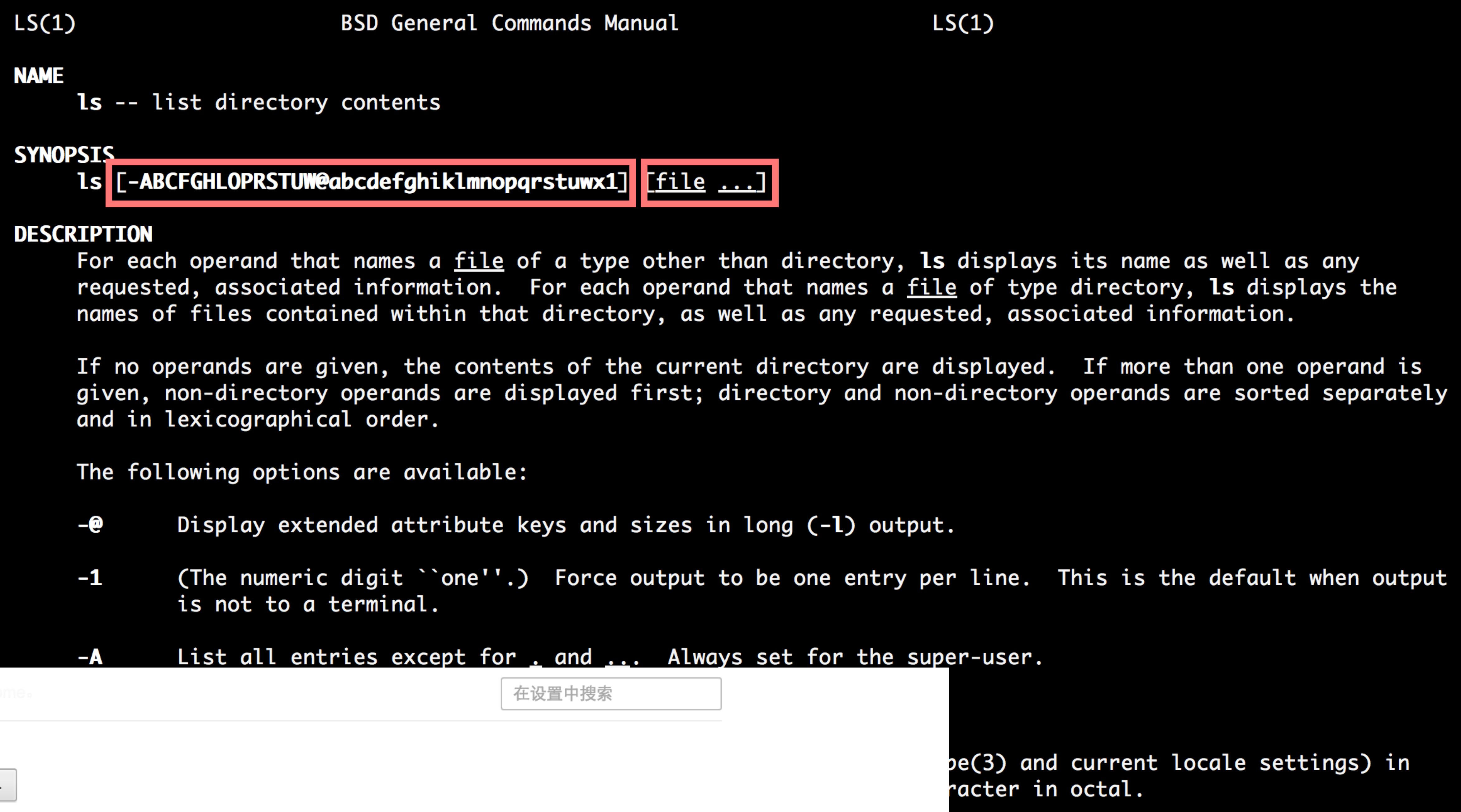
- A simple Example is a table

p_1	p_2	p_3
0	0	0
0	1	1
1	0	1
1	1	0

2-way coverage

Many applications

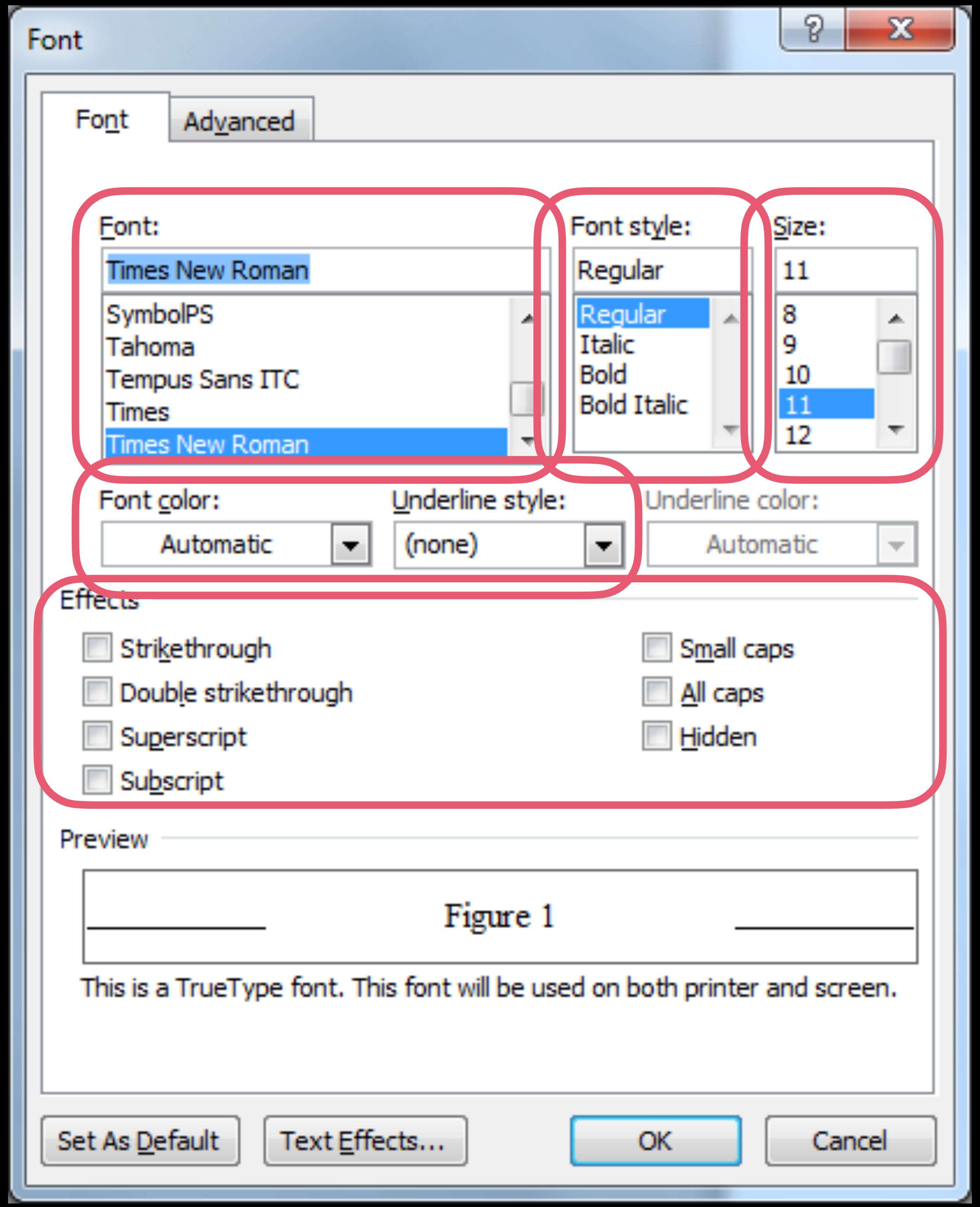




Abstract to Concrete

p_1	p_2	p_3
0	0	0
0	1	1
1	0	1
1	1	0

Abstract



Concrete

Is it enough to detect faults?

南京大學 NANJING UNIVERSITY

Is it enough to detect faults?

p_1	p_2	p_3	Execution Outcome
0	0	0	?
0	1	1	?
1	0	1	?
1	1	0	?

We need Oracle!

- Otherwise, these test cases are meaningless since we do not know whether some of them may trigger failure or not
  - How do we get them?

Common ways in CT

- Assertions, Detailed Specifications (Model-based System, sate transition)
- A correct version as a comparison (Benchmark, e.g. Siemens) , very common in regression testing.
- Trivial ones. e.g., Exceptions, Crashes, etc.

Important, yet not studied in CT

- Oracle is important, but does not attract enough attention in CT
- Either too ideal (full specification, correct version), or too simple (exception)
- Without them, human-based oracle is required, which, is obviously labor-expensive and error-prone.

The target

- We want to make the CT more automatic, in a more general way.
- To reach this target, one inevitable point is to automatically or semi-automatically get an oracle for the generated test case.

One potential solution

- Metamorphic Testing is one of such prominent approach.
- It works when given only some simple properties.

Metamorphic Testing

- Sin(x) function —> Sin(x+360) = Sin(X).
- Hence, when design test inputs, we can have
- 30, 30+360, 30+360+360. They must equal to each other.

Metamorphic Testing

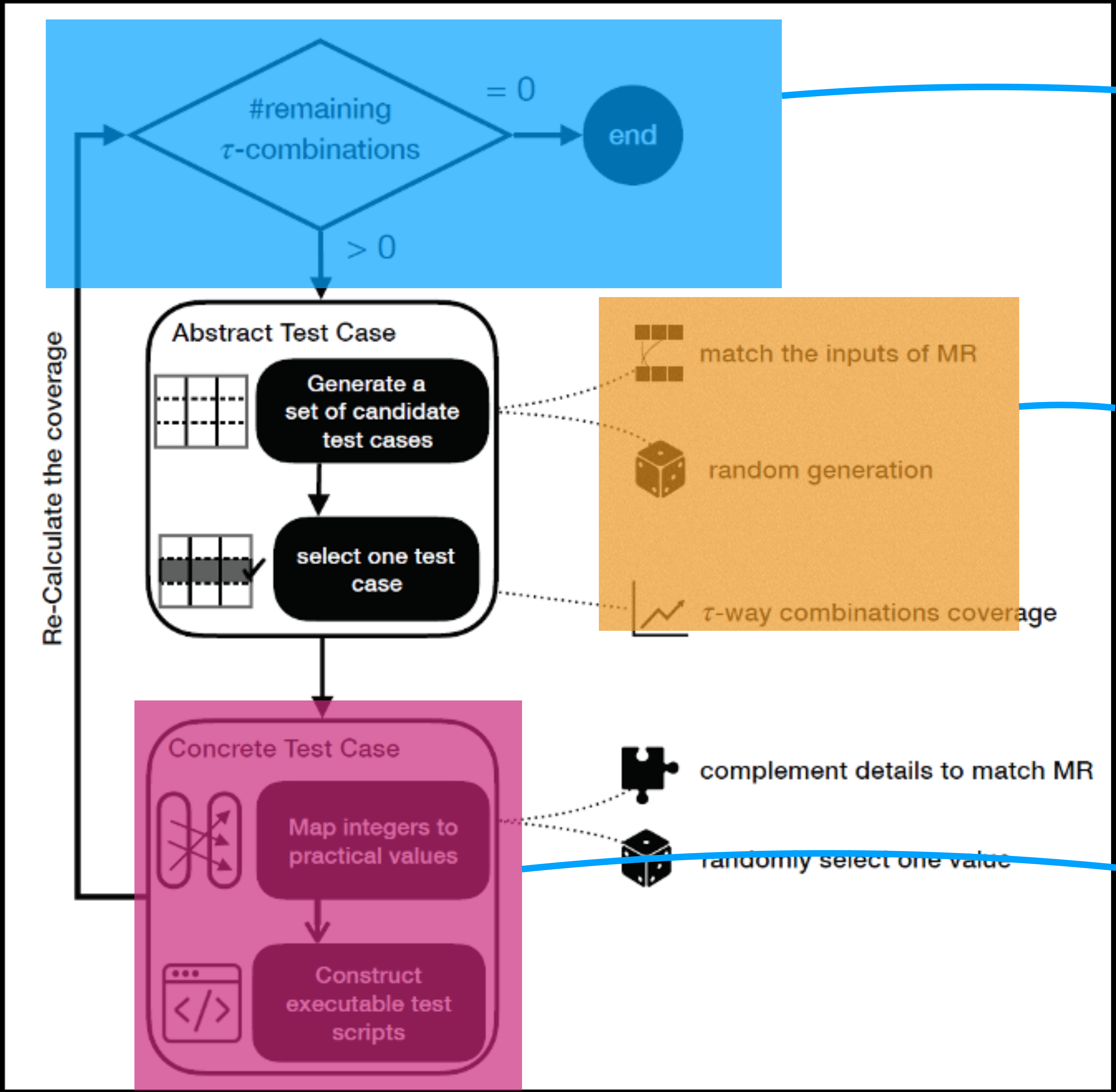
- The key is: Metamorphic Relations
  - Source test and Follow-up test which satisfy MT relationship.

Combine CT with MT?

- It seems that to enhance CT with MT is a good idea, but how to do it?
- Two challenges:
    - CT and MT are both test generation approach, how to generate test cases satisfy both t-way coverage and metamorphic relation relationship?
    - Existing CT generation algorithm are highly optimized for t-way coverage (as diverse as possible), taking metamorphic relationship (multiple test cases share some similarities) into account will do harm to the optimization.

南京大學
NANJING UNIVERSITY

Our approach: COMER



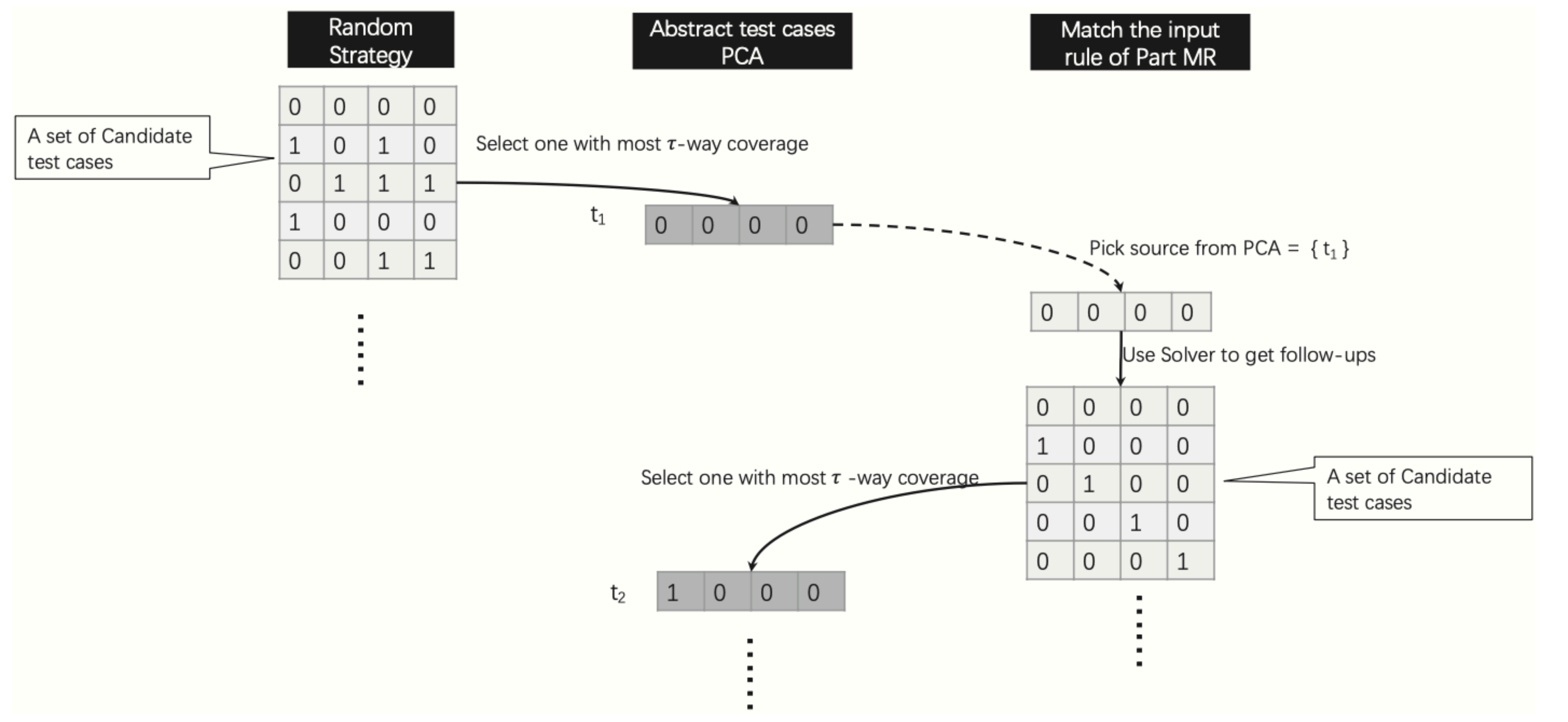
T-way Coverage Satisification part

1. random sampling to get diverse test cases (t-way optimization)

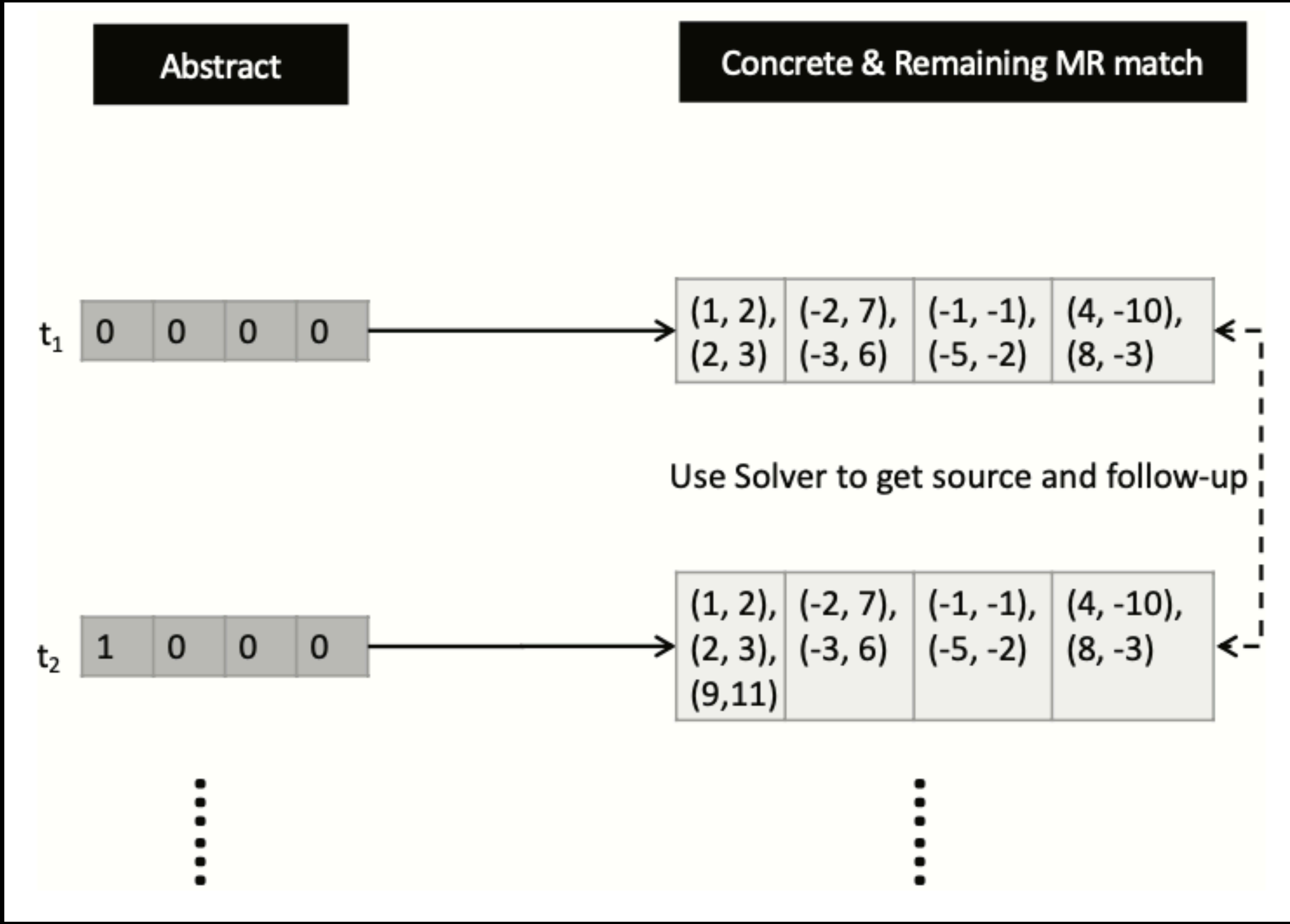
2. Getting chance to give up random sampling, instead, to match source-follow-ups using solver (metamorphic relation)

abstract values to concrete values

Example


Random Strategy
Abstract test cases PCA
Match the input rule of Part MR
A set of Candidate test cases
0 0 0 0
1 0 1 0
0 1 1 1
1 0 0 0
0 0 1 1
Select one with most τ-way coverage
t1
0 0 0 0
Pick source from PCA = { t1 }
0 0 0 0
Use Solver to get follow-ups
0 0 0 0
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
A set of Candidate test cases
Select one with most τ -way coverage
t2
1 0 0 0

Example



Software: Close Pair

Evaluation

- Subjects selection (49 papers 108 programs -> 73 runnable -> 55 satisfied programs).
- Subjects modeling (abstract inputs -> concrete inputs).
- Subjects running scripts (build c++ scripts to run the given program under an abstract inputs).
- Metamorphic Relations Obtaining (For each subject, analyze and verify the metamorphic relation).
- Metamorphic Relation Matching (For any two tests, counting and recording the number relations they have matched).
- Apart from real faults (and we detected faults that are previously not discovered), we use also use Mutation Testing Techniques to mutate the source program, such that we can evaluate the error detection

THE SOFTWARE SUBJECTS UNDER EVALUATION

Software	*Description*	*LOC*	*Abstract IPM*	*Constraints*	*#MRs*	*Faults*
Schedule [41], [42]	Priority scheduler (Siemens suite)	368	$2^2 3^7 8^2$	-	2	real#1
Determinant1 [43]	Matrix determinant computation	98251	$2^1 3^1 5^2$	-	2	novel#1
JAMA [43]	Matrix determinant computation	2858	$2^1 3^1 5^2$	-	2	novel#1
ClosestPair [44]	Finding the closest pair of points	320	$2^1 4^3$	3^1	1	novel#1
Printtoken [42], [45]	Lexical analyzer (Siemens suite)	563	$2^2 3^1 4^4 5^1 10^1 13^2$	4^2	3	real#1
Printtokens2 [42], [45]	Lexical analyzer (Siemens suite)	355	$2^2 3^1 4^4 5^1 10^1 13^2$	4^2	3	seed#10
TCAS [40]	Traffic collision avoidance system	135	$2^7 3^2 4^1 10^2$	-	4	seed#10
F-oneway [46]	Calculate the variance of a single factor	1861	$2^3 3^4$	-	1	novel#1
Multi-MAXSUM [47]	Multi-Segment MAXSUM Algorithm	22	$2^1 3^4 4^1$	$2^2 4^2$	2	seed#9
SurroundedRegion [47]	Capture all regions of a board surrounded by a symbol	59	$2^1 3^2 5^2$	3^3	2	seed#12
MaxRectangle [47]	Find the largest rectangle in a 2D binary matrix	62	$2^1 3^2 5^2$	3^3	2	seed#30
InterleavingString [47]	Decide a string is the interleaving of other two strings	12	$2^1 3^2 5^4$	2^2	1	seed#6
QuickSort [47]	Quick sort algorithm	40	3^6	$2^2 3^1 4^1$	3	seed#5
Bsearch [48]	Binary search within a sorted array	37	$3^4 5^1$	$3^1 4^2 2^2$	2	seed#5
Spwiki [49]	Shortest path between between two vertices in a graph	52	$3^4 4^1$	$2^1 3^1$	2	seed#17
DistinctSubsequence [47]	Count the distinct subsequences of an string	14	$2^1 4^2 5^2$	2^2	1	seed#22
Editingdistance [47]	Enhanced edit distance algorithm	24	$3^2 5^4$	2^2	1	seed#14
FirstMissingPositive [47]	Find the first missing positive integer	13	$2^1 3^5$	$2^2 3^1 4^1$	2	seed#17
HeapSort [47]	Heap sort algorithm	42	3^6	$2^2 3^1 4^1$	3	seed#23
Schedule2 [41], [42]	Priority scheduler (Siemens suite)	347	$2^2 3^7 8^2$	-	2	seed#10
Maxsub [47]	Kadane's MAXSUB algorithm	13	3^5	$2^2 3^1 4^1$	1	seed#6
Jodatime [50]	Date and time utilities	31909	$3^1 4^1 5^6 6^2$	$2^1 3^2$	1	seed#28
Klp [51]	Key-lock problem algorithm	71	$2^2 3^2$	-	2	seed#30
Trisquarej [52]	Returns the type and square of a triangle	70	$3^3 4^1$	$3^1 4^4$	4	seed#30
Boyer [47]	Get the first occurrence of a pattern within a text	248	$2^1 3^6 4^1$	3^1	2	seed#14
Lucene [53]	Text search engine library	19205	$3^6 4^2 5^1$	3^1	3	seed#4
Superstring [54]	Find the shortest common string	61	$2^1 3^1 5^4$	2^2	1	seed#2
Getmid [55]	Compute the median of three integers	19	$2^1 3^3 4^1$	$2^3 3^2$	1	seed#6
RSA [56]	RSA encryption program	11	$3^1 4^2 12^1$	-	1	seed#4
Shortest-path [46]	Get the shortest distance between nodes of the graph	234	$2^1 3^1 4^2 6^1$	-	1	real#1
Rotate [46]	Rotate the matrix	256	$3^1 4^1 6^2$	-	1	novel#1
Argus [46]	Cumulative distribution function of the argus function	1557	$4^1 5^3$	-	1	novel#1

A small example— Grep

Abstract input:

pat_question: [none, begin, middle, end]
pat_a: [none, begin, middle, end]
pat_dash: [none, begin, middle, end]
pat_negate: [none, begin, middle, end]
pat_att: [none, begin, middle, end]
pat_ato: [none, begin, middle, end]
pat_questionStar: [none, begin, middle, end]
pat_aStar: [none, begin, middle, end]
pat_dashStar: [none, begin, middle, end]
pat_negateStar: [none, begin, middle, end]
pat_attStar: [none, begin, middle, end]
pat_atoStar: [none, begin, middle, end]
pat_bol: [off, on]
pat_eol: [off, on]
pat_atn: [off, on]
pat_at: [off, on]
pat_bracket:[[?-?],[*],[?/l…/l?], [:lower:]]
bracket _attribute: [non

A small example— Grep

- Concrete input
  - Grep [0-9][a-z] test.txt

A small example— Grep

Constraints :

pat_question =begin => pat_a !=begin && pat_dash != begin && pat_negate != begin && pat_att != begin && pat_ato != begin && pat_questionStar != begin && pat_aStar != begin && pat_dashStar != begin && pat_negateStar !=begin && pat_attStar != begin && pat_atoStar != begin && pat_bol !=on && bracket _attribute != begin
pat_a =begin => pat_question !=begin && pat_dash != begin && pat_negate != begin && pat_att != begin && pat_ato != begin && pat_questionStar != begin && pat_aStar != begin && pat_dashStar != begin && pat_negateStar !=begin && pat_attStar != begin && pat_atoStar != begin && pat_bol != on&& bracket _attribute != begin
pat_dash = begin => pat_a !=begin && pat_question !=begin && pat_negate != begin && pat_att != begin && pat_ato != begin && pat_questionStar != begin && pat_aStar != begin && pat_dashStar != begin && pat_negateStar !=begin && pat_attStar != begin && pat_atoStar != begin && pat_bol !=on&& bracket _attribute != begin
pat_negate = begin =>pat_dash != begin && pat_a !=begin && pat_question !=begin && pat_att != begin && pat_ato != begin && pat_questionStar != begin && pat_aStar != begin && pat_dashStar != begin && pat_negateStar !=begin && pat_attStar != begin && pat_atoStar != begin && pat_bol !=on&& bracket _attribute != begin
pat_att = begin => pat_negate != begin && pat_dash != begin && pat_a !=begin && pat_question !=begin && pat_ato != begin && pat_questionStar != begin && pat_aStar != begin && pat_dashStar != begin && pat_negateStar !=begin && pat_attStar != begin && pat_atoStar != begin && pat_bol != bracket _attribute != begin
pat_ato = begin => pat_att != begin

A small example— Grep

- MR relationships
  - mr0: 测试用例1为…[?-?]…,　测试用例2为…[*]…。
    - 如[a-d]和[abcd]。
  - mr1: 测试用例1为…[?-?]…,　测试用例2为…[?/|…/|?]…。
    - 如[a-d]和[a/|b/|c/|d]。
  - mr2:　测试用例1为…[*]…,　　测试用例2为…[?/|…/|?]…。
    - 如[abcd]和[a/|b/|c/|d]。

A small example— Grep

commonmath.cpp (~/Desktop/MR/code/commonmath) - VIM

```cpp
#include <iostream>
#include <string>
#include <cstdlib>
#include <stdio.h>
#include <stdlib.h>
#include <cmath>
#include <climits>
#include <ctime>

int main(int argc, char *argv[])
{
        if (argc < 5)
        {
                std::cerr << "bad args" << std::endl;
        }
        int arg1 = atoi(argv[1]);
        int arg2 = atoi(argv[2]);
        int arg3 = atoi(argv[3]);
        int arg4 = atoi(argv[4]);
        int arg5 = atoi(argv[5]);

    srand((unsigned)time(NULL));

    int row,column;

    switch (arg1)
        {
                case 0: row=1+rand()%9;
                        break;
                case 1: row=10+rand()%10;
                                break;
                case 2: row=20+rand()%10;
                                break;
        }

    switch (arg2)
        {
                case 0: column=1+rand()%9;
                        break;
                case 1: column=10+rand()%10;
                                break;
                case 2: column=20+rand()%10;
                                break;
        }

        if(arg3==2||arg3==3||arg3==4)
        column=row;

        int n=row*column;
    int a[500][500];
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<column;j++)
        a[i][j]=-1;
    }
```

judgemr.cpp (~/Desktop/MR/code/commonmath) - VIM

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <cmath>
#include <dirent.h>
using namespace std;
double b[500][500];
double c[500][500][500];
double y[500];

int bi;
int judge[500];   //judge-
int judge1[500];  //judge.

int result[500][5];
int resulti=0;

void spl(string a,int count)  //½«a»®·ÖÎªÕûÊýÊý×é
{
        for(int i=0;i<500;i++)
        {
                judge[i]=0;
                judge1[i]=-1;
        }
        int b0=0;
    string temp[500];
    for(int i=0;i<500;i++)
    temp[i]="";
    int tempi=0;
    int judge1i=0;
    for(int i=0;i<a.length();i++)
    {
        if((a[i]>='0'&&a[i]<='9')||a[i]=='-'||a[i]=='.')
        {
                temp[tempi]+=a[i];
                if(a[i]=='-')
                judge[tempi]=1;
                if(a[i]=='.')
                judge1[tempi]=judge1i;
                judge1i++;
        }
        if(a[i]==' ')
        {
                tempi++;
                judge1i=0;
        }
        if(i==a.length()-1&&a[i]!=' ')
            tempi++;
    }

    for(int i=0;i<tempi;i++)
    {
         double tempint=0;
         int cheng=1;
```

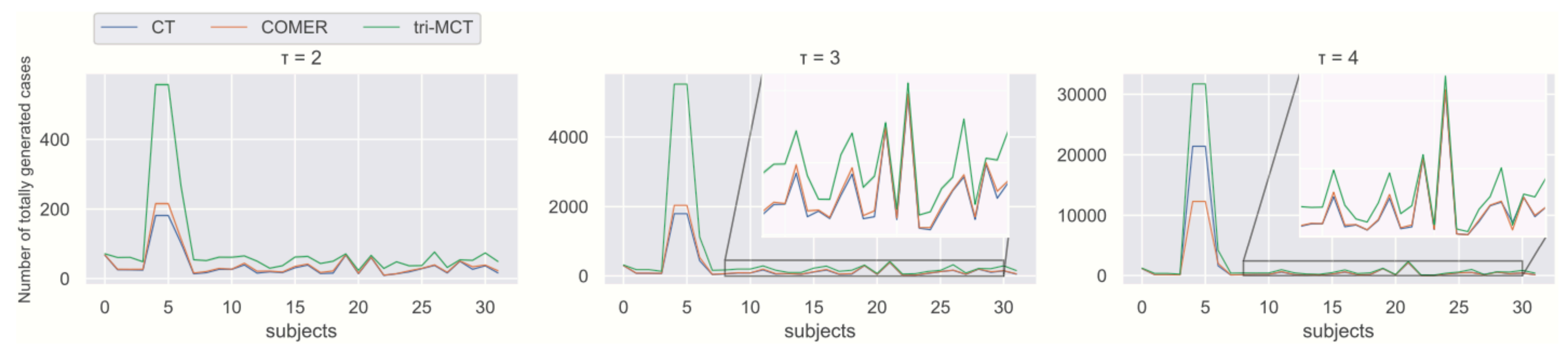
Research Question

- Is COMER effective and efficient at handling the automated oracle problem?
- Compared with using optimal oracles, how does COMER lose in fault detection by the mere use of MR
- What features of the metamorphic relations affect the performance of COMER

RQ1

- Comparison Approach
  - Pure CT
  - Trivially first using CT to generate test cases, and then for each test case , regard it as a source, then generate a follow-up
- Metric:
  - Number of test cases
  - matchings of sources and follow-ups
  - detected faults

Results



COMER and pure CT are similar (CT is slightly better), the last is tri-MCT

南京大学 NANJING UNIVERSITY

Results

Fig. 2. The matchings of Source&Follow-ups by CT, COMER, and tri-MCT

tri-MCT is the best, then COMER, while the last is pure CT (which is hardly to match source and follow-up)

Results

Finding1
COMER is effective at improving the number of matchings of Source&Follow-ups and the fault detection rate, while remaining a relatively small testing cost.

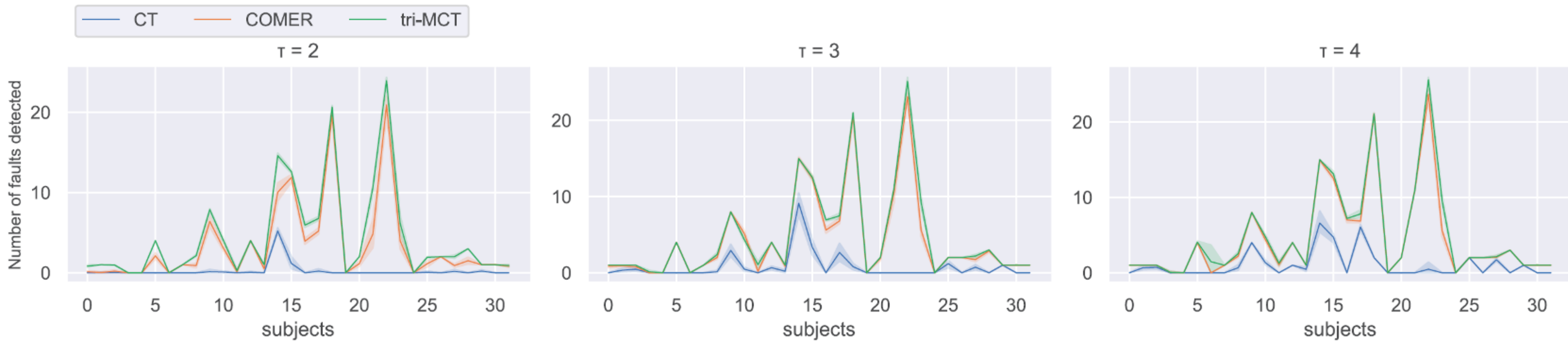


Fig. 3. The number of faults detected by CT, COMER, and tri-MCT

Similar fault detection between COMER and tri-MCT, both better than pure CT.

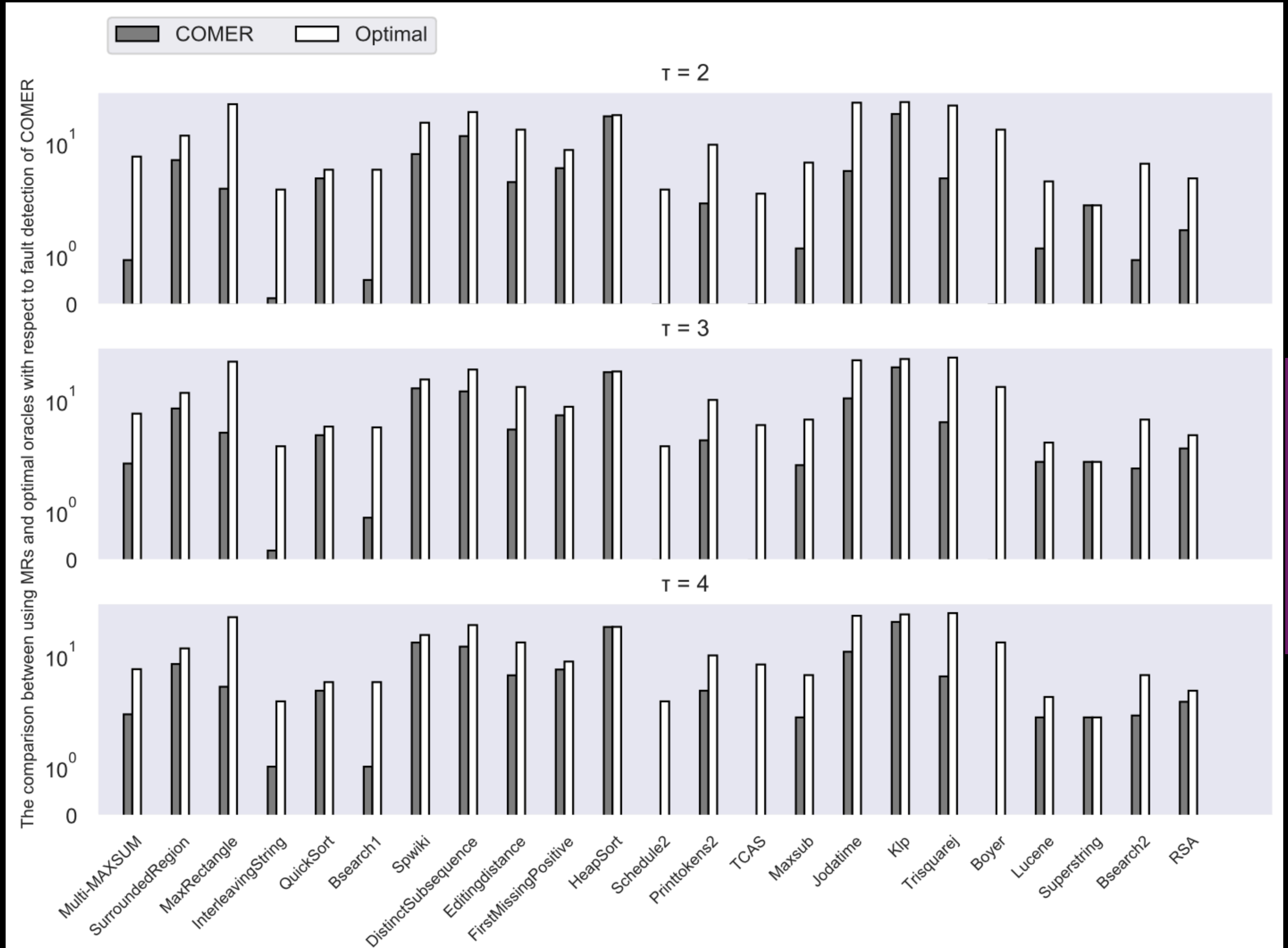
RQ2

- Compared with using optimal oracles, how does COMER lose in fault detection by the mere use of MR

- In order to give such an *optimal* oracle, we need to utilize a completely correct version of the subject under testing. After that, we can tell the *pass* or *fail* for a test case of a faulty version by checking whether the outcome of this test case is equal to that of the correct version.

南京大學
NANJING UNIVERSITY

Results



Finding 2:
By merely utilizing metamorphic relation, COMER achieved about a 42% fault detection rate when compared with using optimal oracles. The number of detected faults varies among subjects but remains stable when the testing strength is larger than 2

RQ3

- What features of the metamorphic relations affect the performance of COMER

THE INVESTIGATED FEATURES OF METAMORPHIC RELATIONS

Feature	Short Description
Source Generate	Given one MR, the percentage of the test cases that can be treated as source test cases among all the possible test cases.
Follow Generate	Given one MR, the percentage of the test cases that can be treated as follow-up test cases among all the possible test cases.
Output Match	Given one MR, the degree of the difficulty that its output rule can be satisfied. In our experiments, the degrees of the difficulty are classified into 5 main levels (from easy to difficult): 1) The "unequal" relation between two outputs with single values 2) The "equal" relation between two outputs with single limited values (e.g., enumerated type) 3) The "equal" relation between two outputs with a set of limited values 4) The "equal" relation between two outputs with single unlimited values (e.g., float number) 5) The "equal" relation between two outputs with a set of unlimited values

Results

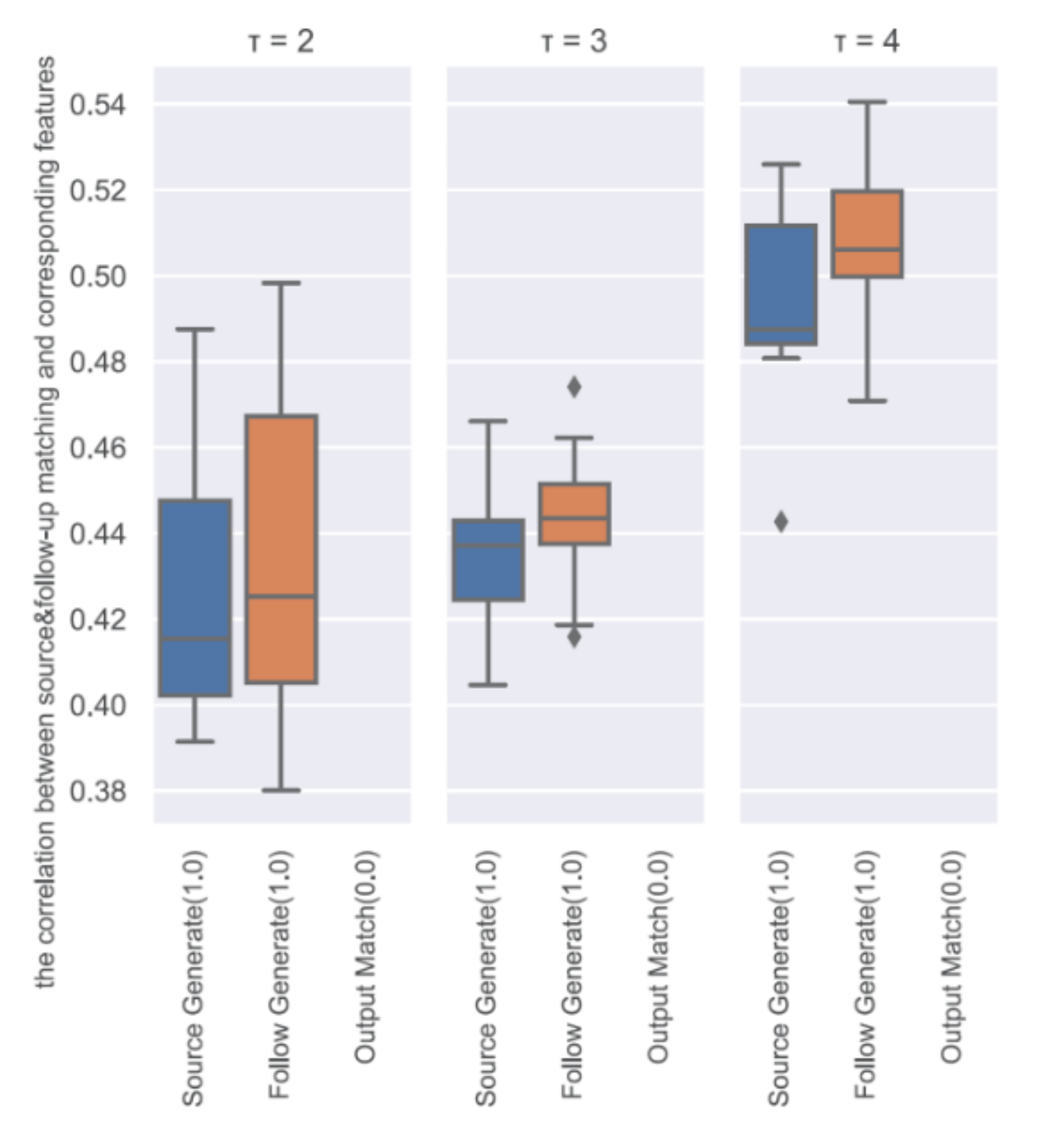
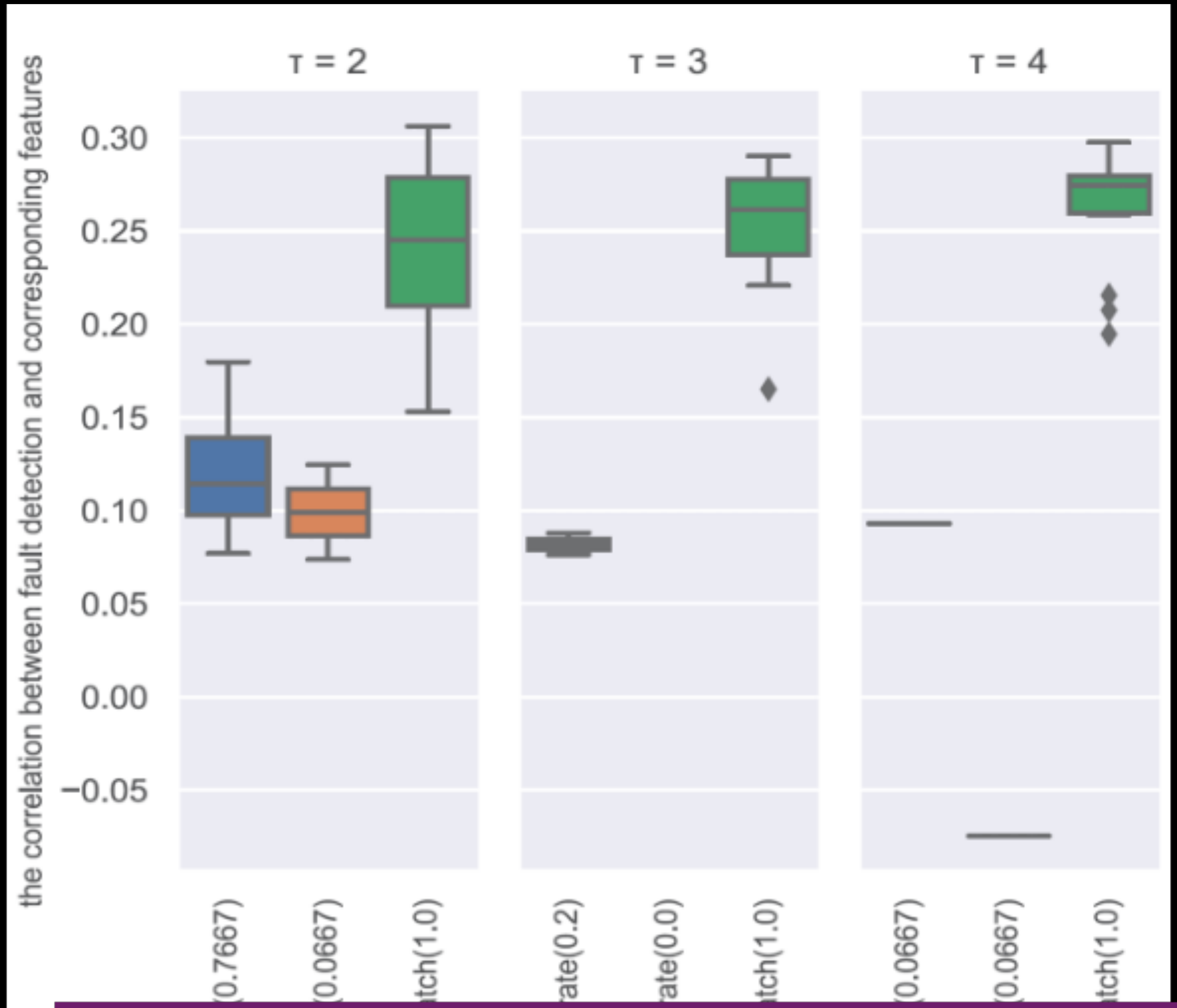


Fig. 5. The correlation between different features and the matchings of sources and follow-ups



Fi

Finding3
The degree of the difficulty that the input rules of a MR can be satisfied is moderately correlated to the performance of COMER in terms of the number of Source&Follow-up matchings, while the degree of the difficulty that the output rules of a MR can be satisfied is modestly correlated to the number of detected faults.

Summary

- Oracle is one issue to get CT fully automated
- This report presents COMER, an approach combines CT and MT
  - The outline is t-way coverage satisification using random sampling
  - Give chances to match source and follow-up test cases
- Experiments on 31 subjects shows the efficacy of COMER.
  - The properties of MR affect the performance of COMER
  - Only using metamorphic testing is still far from optimal

Thanks!
Q&A

xintao niu
niuxintao@nju.edu.cn

智能软件与工程学院
School of Intelligent Software and Engineering