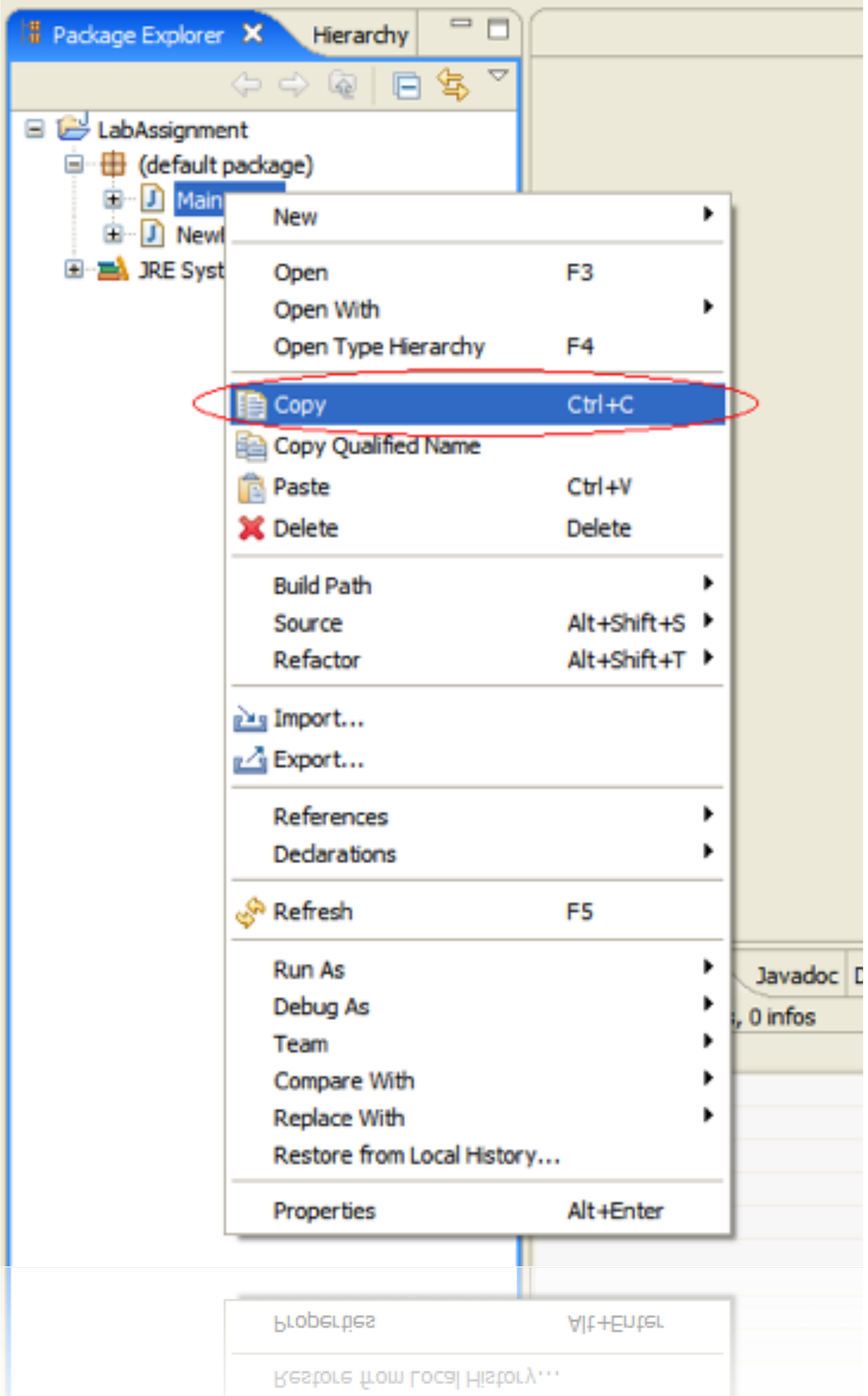
Similarity of Source Code in the Presence of Pervasive Modifications

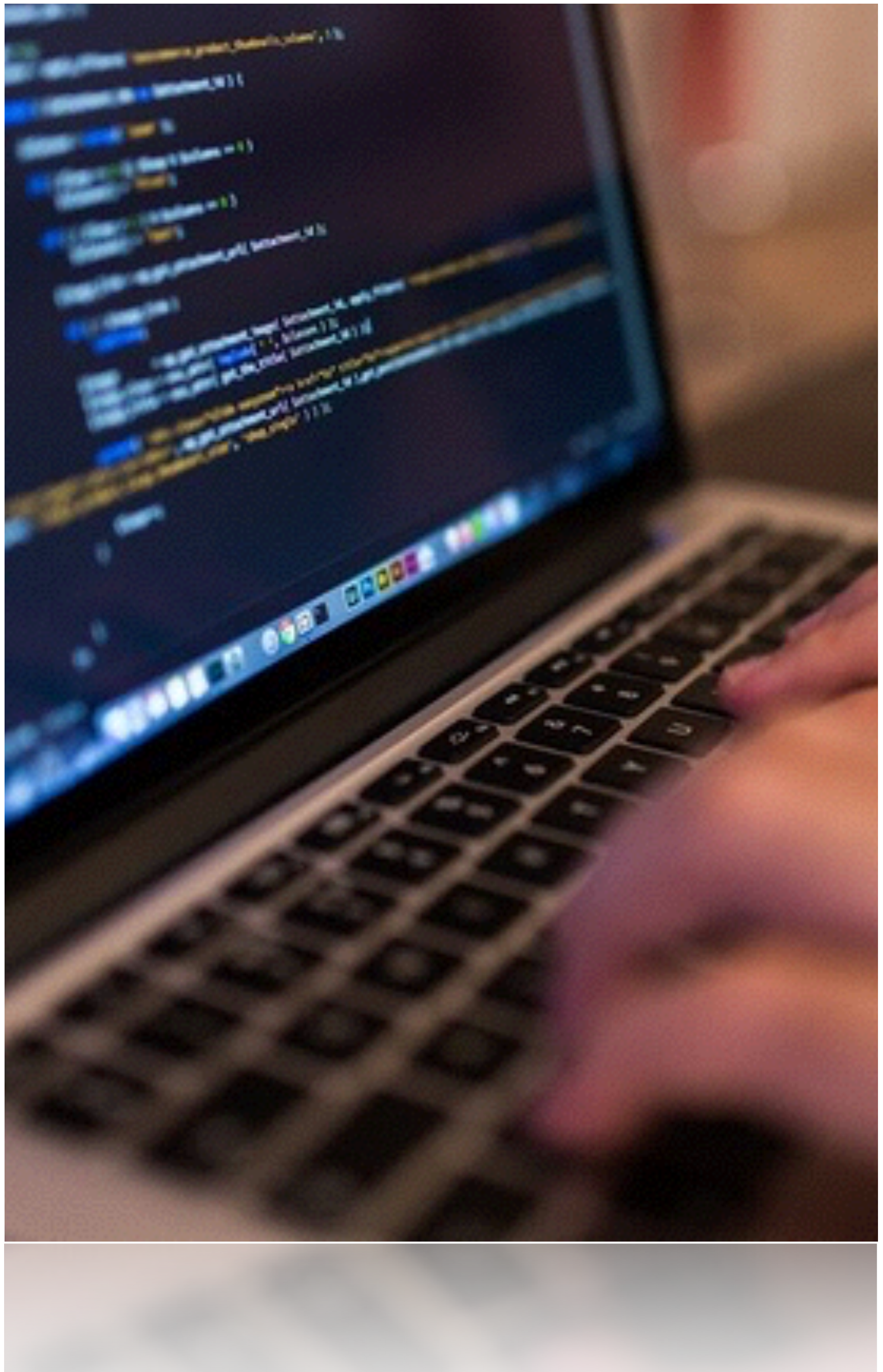
Chaiyong Ragkhitwetsagul, Jens Krinke, David Clark
CREST, Department of Computer Science
University College London, UK

Measuring Similarity of Source Code



Locating duplicated code fragment (clone detection)

Measuring Similarity of Source Code

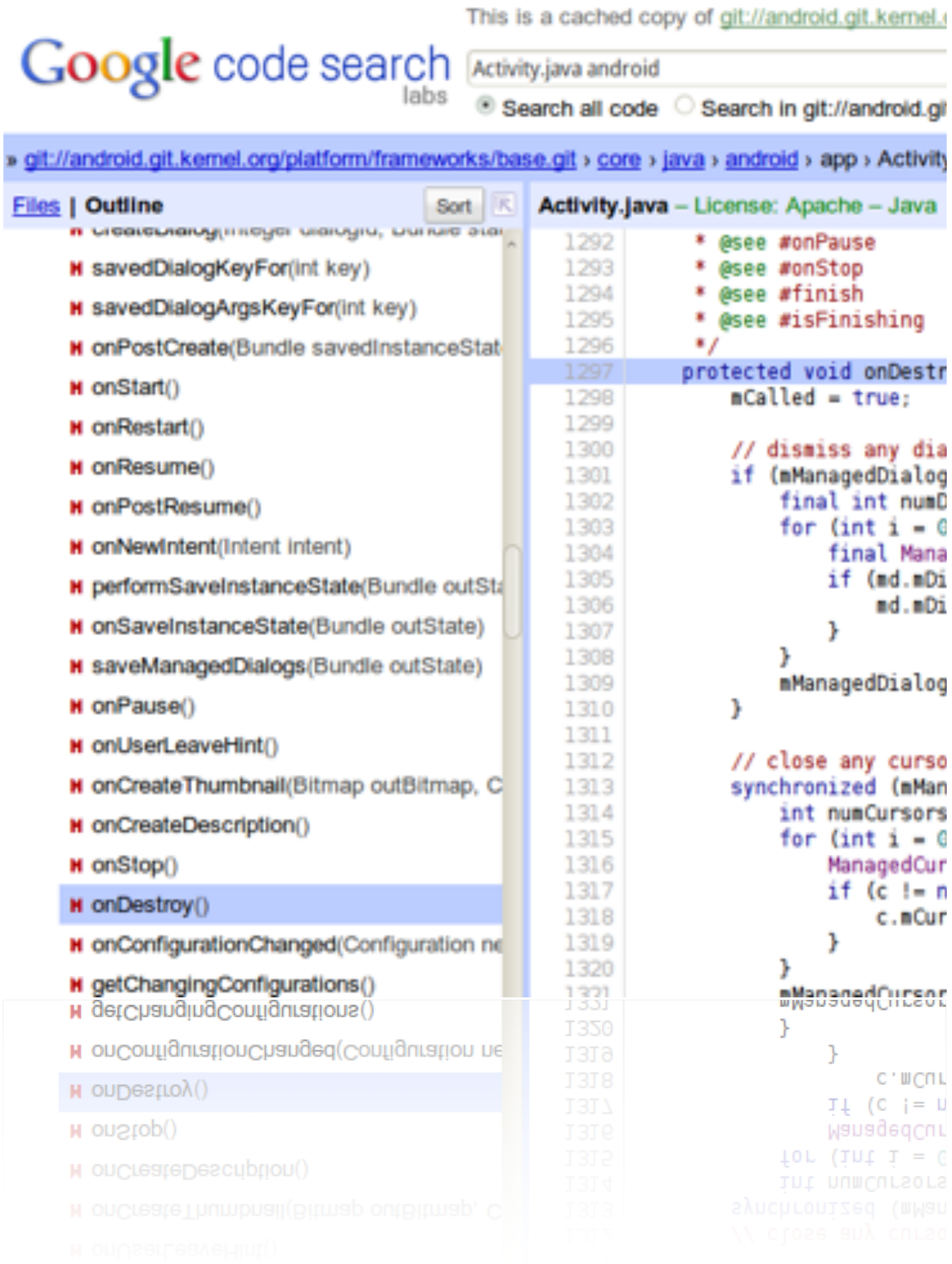


Locating duplicated code fragment (clone detection)

Plagiarism detection

Software copyright infringement

Measuring Similarity of Source Code



Google code search labs

This is a cached copy of git://android.git.kernel.

Activity.java android

● Search all code ○ Search in git://android.gi

» git://android.git.kernel.org/platform/frameworks/base.git › core › java › android › app › Activity

Files | Outline Sort

Activity.java – License: Apache – Java

createDialog(Integer dialogId, Bundle stat
savedDialogKeyFor(int key)
savedDialogArgsKeyFor(int key)
onPostCreate(Bundle savedInstanceStat
onStart()
onRestart()
onResume()
onPostResume()
onNewIntent(Intent intent)
performSaveInstanceState(Bundle outSta
onSaveInstanceState(Bundle outState)
saveManagedDialogs(Bundle outState)
onPause()
onUserLeaveHint()
onCreateThumbnail(Bitmap outBitmap, C
onCreateDescription()
onStop()
onDestroy()
onConfigurationChanged(Configuration ne
getChangingConfigurations()

```
     * @see #onPause
     * @see #onStop
     * @see #finish
     * @see #isFinishing
     */
    protected void onDestr
        mCalled = true;

        // dismiss any dia
        if (mManagedDialog
            final int numD
            for (int i = 0
                final Mana
                if (md.mDi
                    md.mDi
                }
            }
            mManagedDialog
        }

        // close any curso
        synchronized (mMan
            int numCursors
            for (int i = 0
                ManagedCur
                if (c != n
                    c.mCur
                }
            }
            mManagedCursor
```

Locating duplicated code fragment (clone detection)

Plagiarism detection

Software copyright infringement

Code search

finding similar bug fixes, program comprehension, code recommendation, and example extraction

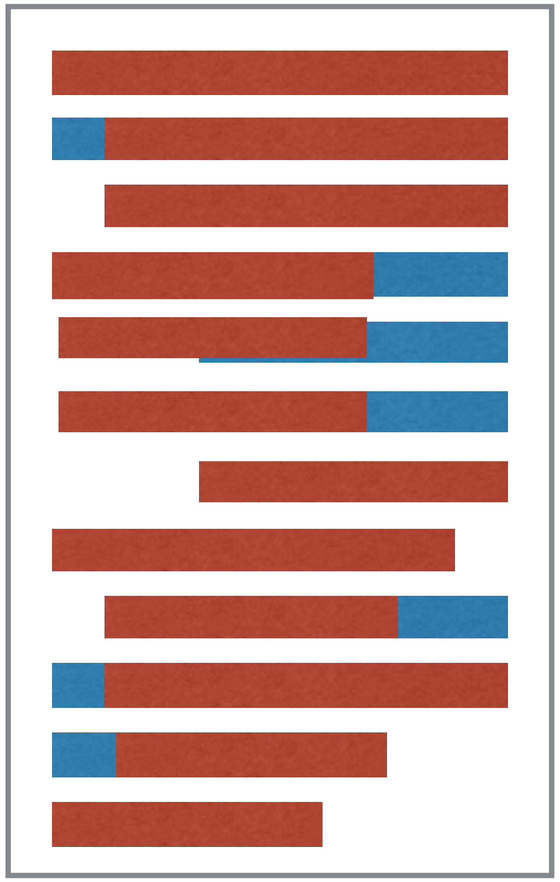
Pervasive Modifications

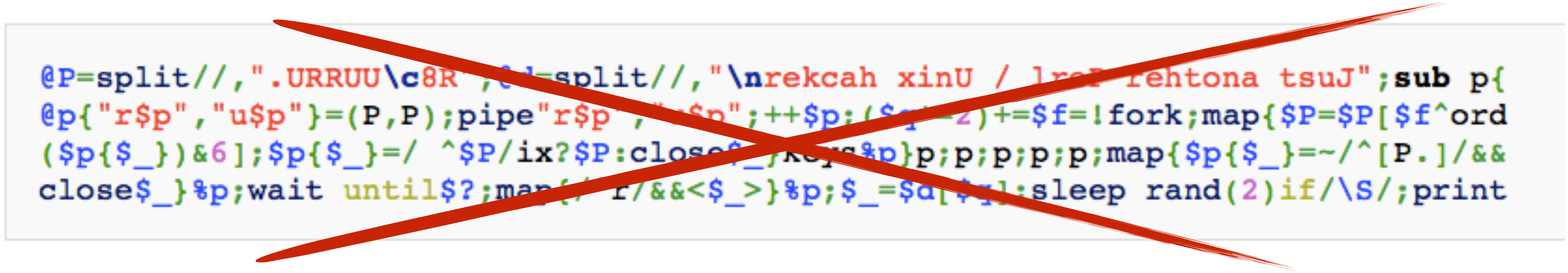
Changes in layout or renaming of identifiers

Changes that affect the code **globally.**

Normally found in code cloning, software plagiarism, software evolution

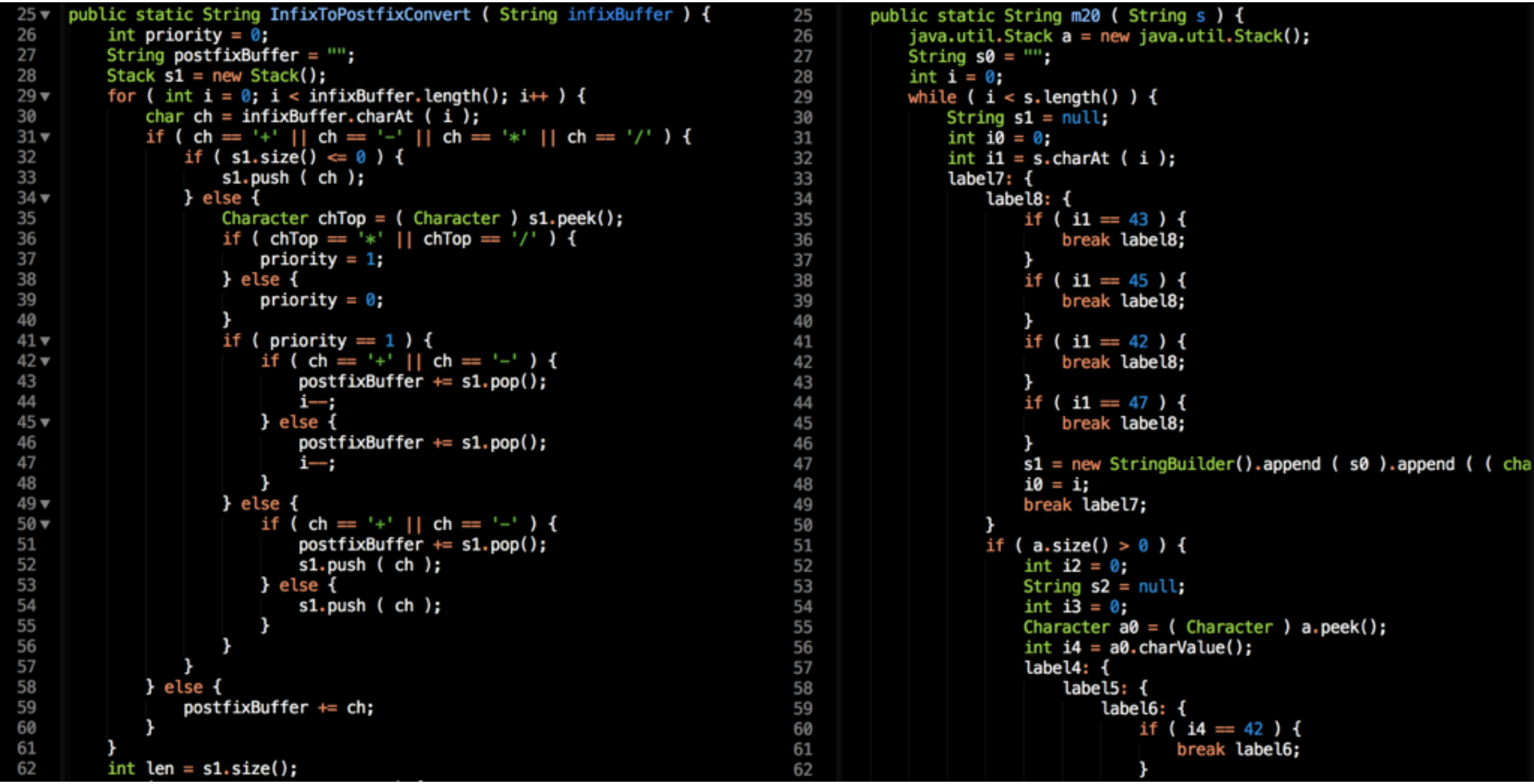
Not include code **obfuscation modifications**





```
@P=split//,".URRUU\c8R";@d=split//,"\nrekcah xinU / lreP rehtona tsuJ";sub p{
@p{"r$p","u$p"}=(P,P);pipe"r$p","u$p";++$p;($q*=2)+=$f=!fork;map{$P=$P[$f^ord
($p{$_})&6];$p{$_}=/ ^$P/ix?$P:close$_}keys%p}p;p;p;p;p;map{$p{$_}=~/^[P.]/&&
close$_}%p;wait until$?;map{/^r/&&<$_>}%p;$_=$d[$q];sleep rand(2)if/\S/;print
```

Pervasive Modifications



```
public static String InfixToPostfixConvert ( String infixBuffer ) {
    int priority = 0;
    String postfixBuffer = "";
    Stack s1 = new Stack();
    for ( int i = 0; i < infixBuffer.length(); i++ ) {
        char ch = infixBuffer.charAt ( i );
        if ( ch == '+' || ch == '-' || ch == '*' || ch == '/' ) {
            if ( s1.size() <= 0 ) {
                s1.push ( ch );
            } else {
                Character chTop = ( Character ) s1.peek();
                if ( chTop == '*' || chTop == '/' ) {
                    priority = 1;
                } else {
                    priority = 0;
                }
                if ( priority == 1 ) {
                    if ( ch == '+' || ch == '-' ) {
                        postfixBuffer += s1.pop();
                        i--;
                    } else {
                        postfixBuffer += s1.pop();
                        i--;
                    }
                } else {
                    if ( ch == '+' || ch == '-' ) {
                        postfixBuffer += s1.pop();
                        s1.push ( ch );
                    } else {
                        s1.push ( ch );
                    }
                }
            }
        } else {
            postfixBuffer += ch;
        }
    }
    int len = s1.size();
```

```
public static String m20 ( String s ) {
    java.util.Stack a = new java.util.Stack();
    String s0 = "";
    int i = 0;
    while ( i < s.length() ) {
        String s1 = null;
        int i0 = 0;
        int i1 = s.charAt ( i );
        label7: {
            label8: {
                if ( i1 == 43 ) {
                    break label8;
                }
                if ( i1 == 45 ) {
                    break label8;
                }
                if ( i1 == 42 ) {
                    break label8;
                }
                if ( i1 == 47 ) {
                    break label8;
                }
                s1 = new StringBuilder().append ( s0 ).append ( ( cha
                i0 = i;
                break label7;
            }
            if ( a.size() > 0 ) {
                int i2 = 0;
                String s2 = null;
                int i3 = 0;
                Character a0 = ( Character ) a.peek();
                int i4 = a0.charValue();
                label4: {
                    label5: {
                        label6: {
                            if ( i4 == 42 ) {
                                break label6;
                            }
```

NCD difflib CCFinder Deckard cosine-similarity

iClones fuzzywuzzy jellyfish

When source code is pervasively modified, which similarity detection techniques or tools get the most accurate results?

diff NiCad Sim n-gram

JPlag

bsdiff Simian Plaggie Sherlock

Empirical Study

RQ1 (Performance comparison): How well do current similarity detection techniques perform in the presence of pervasive source code modifications?

Empirical Study

RQ1 (Performance comparison): How well do current similarity detection techniques perform in the presence of pervasive source code modifications?

RQ2 (Optimal configurations): What are the best parameter settings and similarity thresholds for the techniques?

Empirical Study

RQ1 (Performance comparison): How well do current similarity detection techniques perform in the presence of pervasive source code modifications?

RQ2 (Optimal configurations): What are the best parameter settings and similarity thresholds for the techniques?

RQ3 (Normalisation by decompilation): Does use of compilation followed by decompilation as a pre-processing normalisation method improve detection results?

Empirical Study

RQ1 (Performance comparison): How well do current similarity detection techniques perform in the presence of pervasive source code modifications?

RQ2 (Optimal configurations): What are the best parameter settings and similarity thresholds for the techniques?

RQ3 (Normalisation by decompilation): Does use of compilation followed by decompilation as a pre-processing normalisation method improve detection results?

RQ4 (Reuse of configurations): Can we apply the derived optimal configurations for a tool created on one data set to other data sets effectively?

Obfuscators

ARTIFICE*

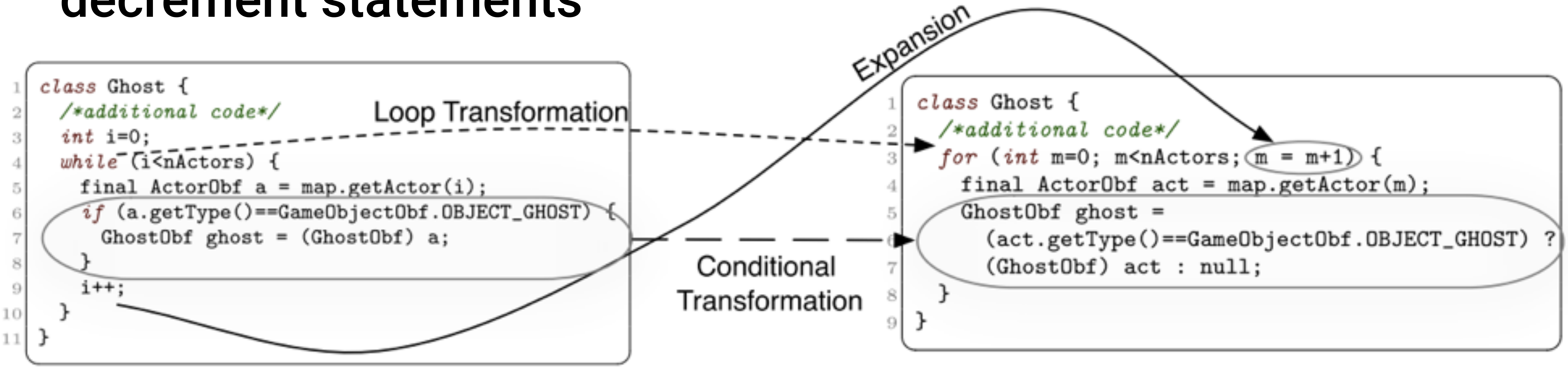
Source code level

Renaming, changing loops & conditional statements, changing increment/ decrement statements

ProGuard

Bytecode level

Rename classes, fields, variables to short, meaningless



```
class Ghost {
  /*additional code*/
  int i=0;
  while (i<nActors) {
    final ActorObf a = map.getActor(i);
    if (a.getType()==GameObjectObf.OBJECT_GHOST) {
      GhostObf ghost = (GhostObf) a;
    }
    i++;
  }
}
```

a) original code fragment

```
class Ghost {
  /*additional code*/
  for (int m=0; m<nActors; m = m+1) {
    final ActorObf act = map.getActor(m);
    GhostObf ghost =
      (act.getType()==GameObjectObf.OBJECT_GHOST) ?
      (GhostObf) act : null;
  }
}
```

b) obfuscated code fragment

* Schulze, S., & Meyer, D. (2013). On the robustness of clone detection to code obfuscation. 2013 7th International Workshop on Software Clones (IWSC)

Experimental Scenarios

Scenario 1

Pervasive Modifications

Scenario 2

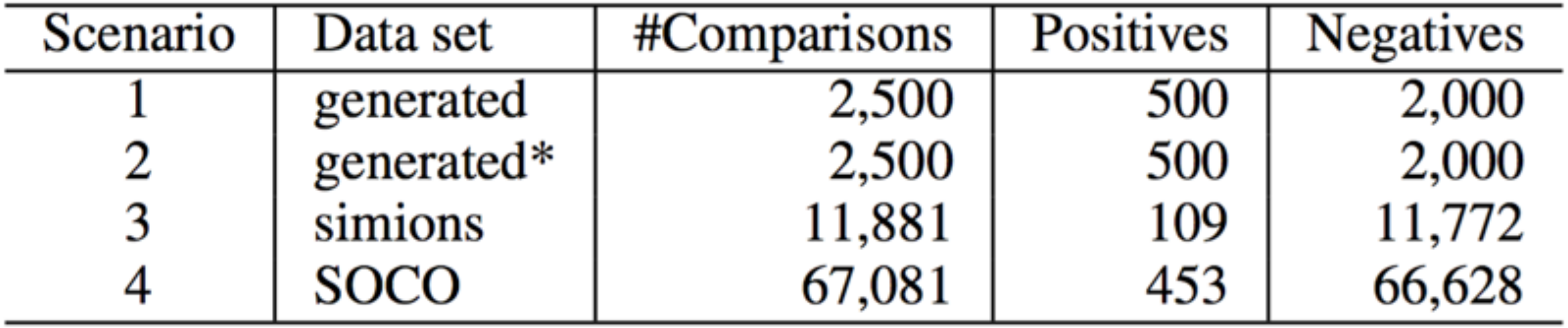
Decompilation

Scenario 3

Semantically Similar Code

Scenario 4

Reused Boiler-plate Code



Scenario	Data set	#Comparisons	Positives	Negatives
1	generated	2,500	500	2,000
2	generated*	2,500	500	2,000
3	simions	11,881	109	11,772
4	SOCO	67,081	453	66,628

Scenario 1

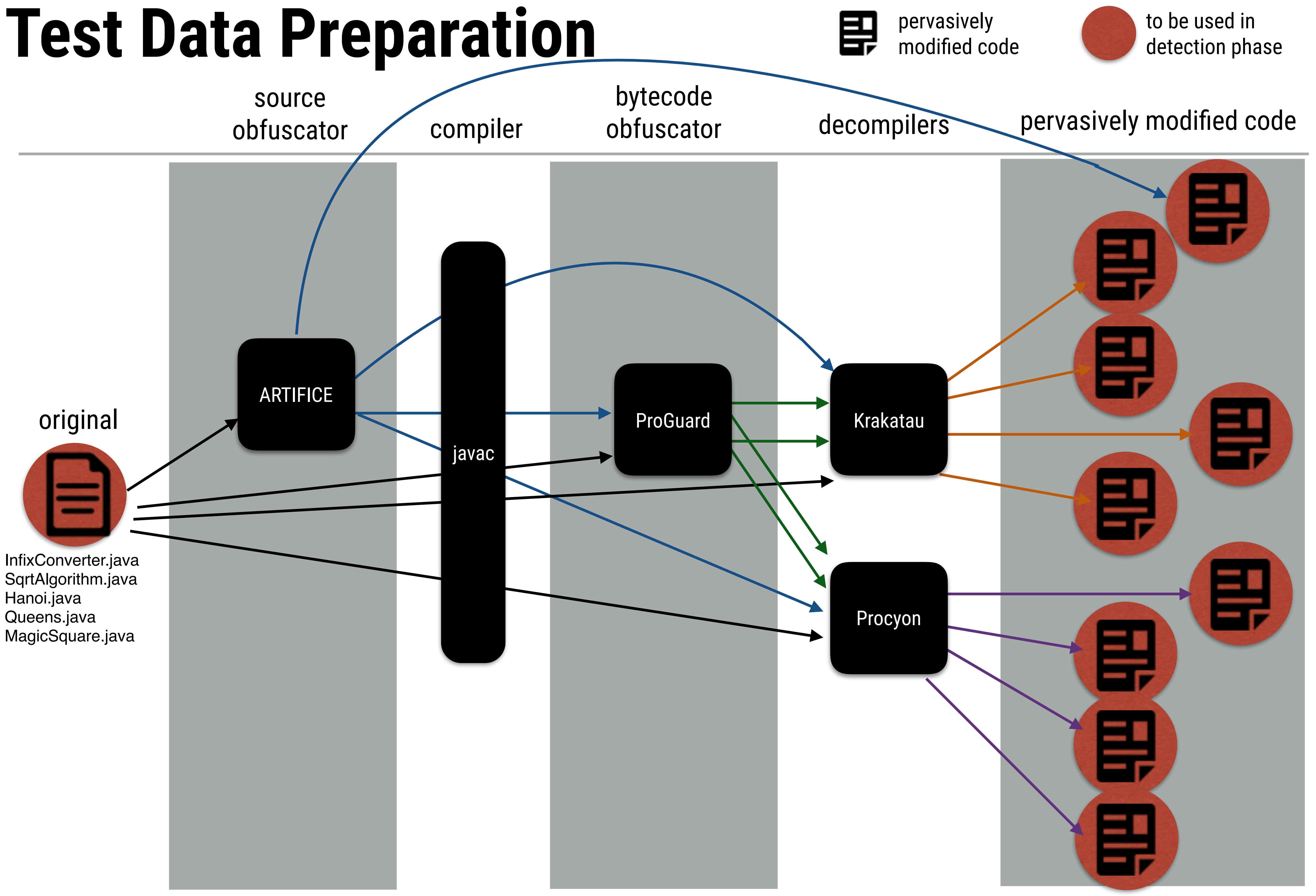
Pervasive Modifications

Studies tool performance against pervasive modifications

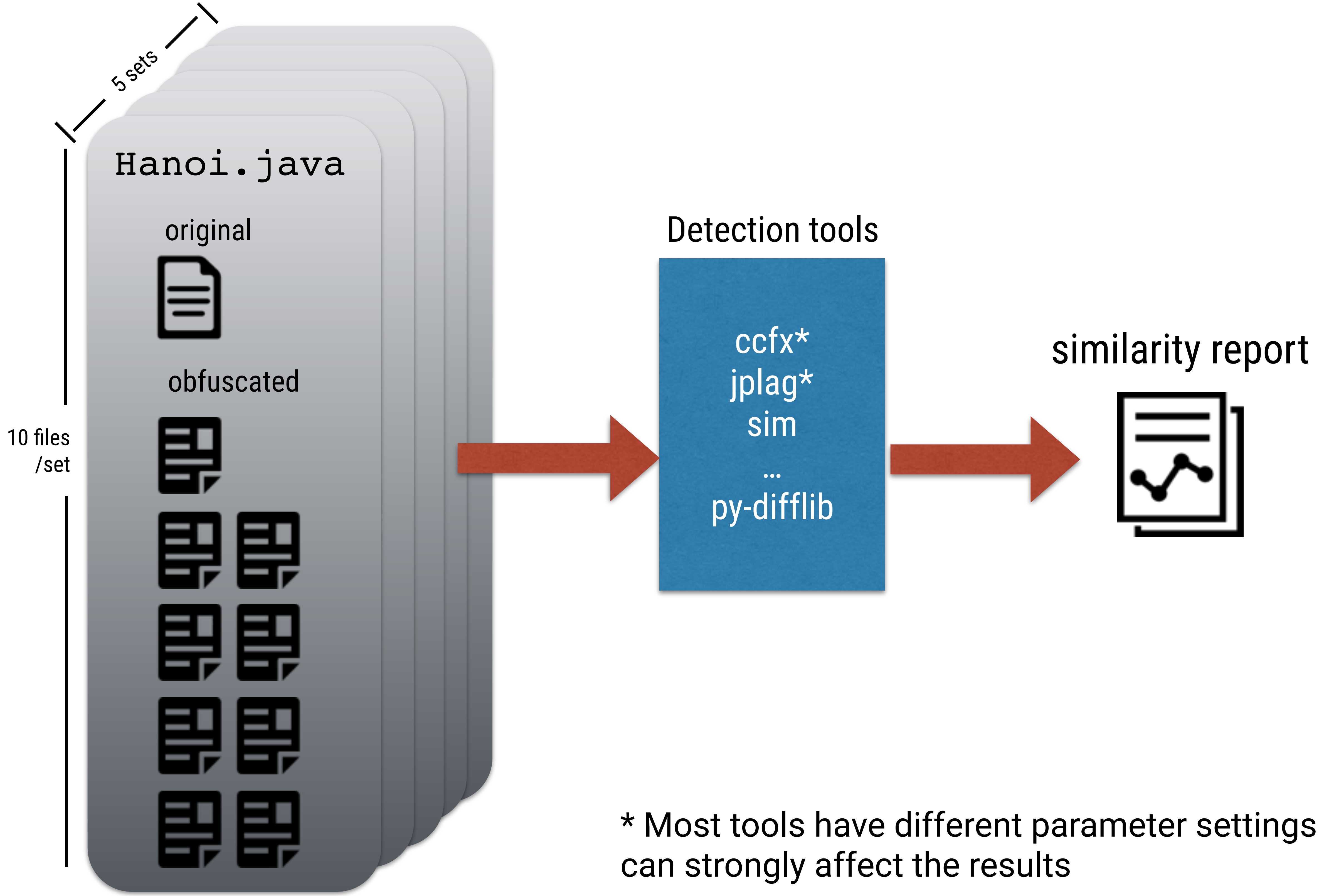
Simulated through source and bytecode obfuscation

The best configuration for every tool is discovered

Test Data Preparation



Similarity Calculation

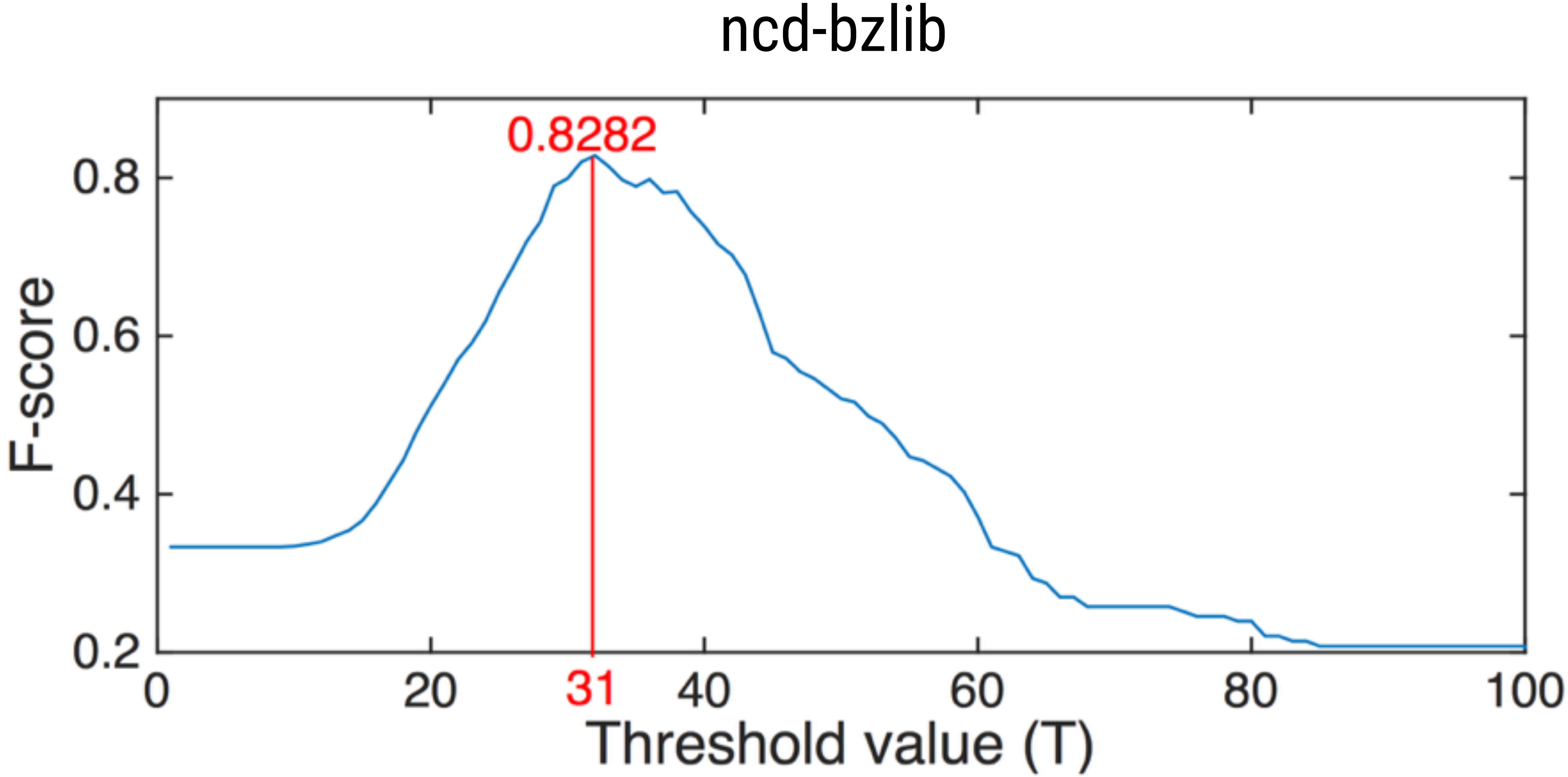


* Most tools have different parameter settings which can strongly affect the results

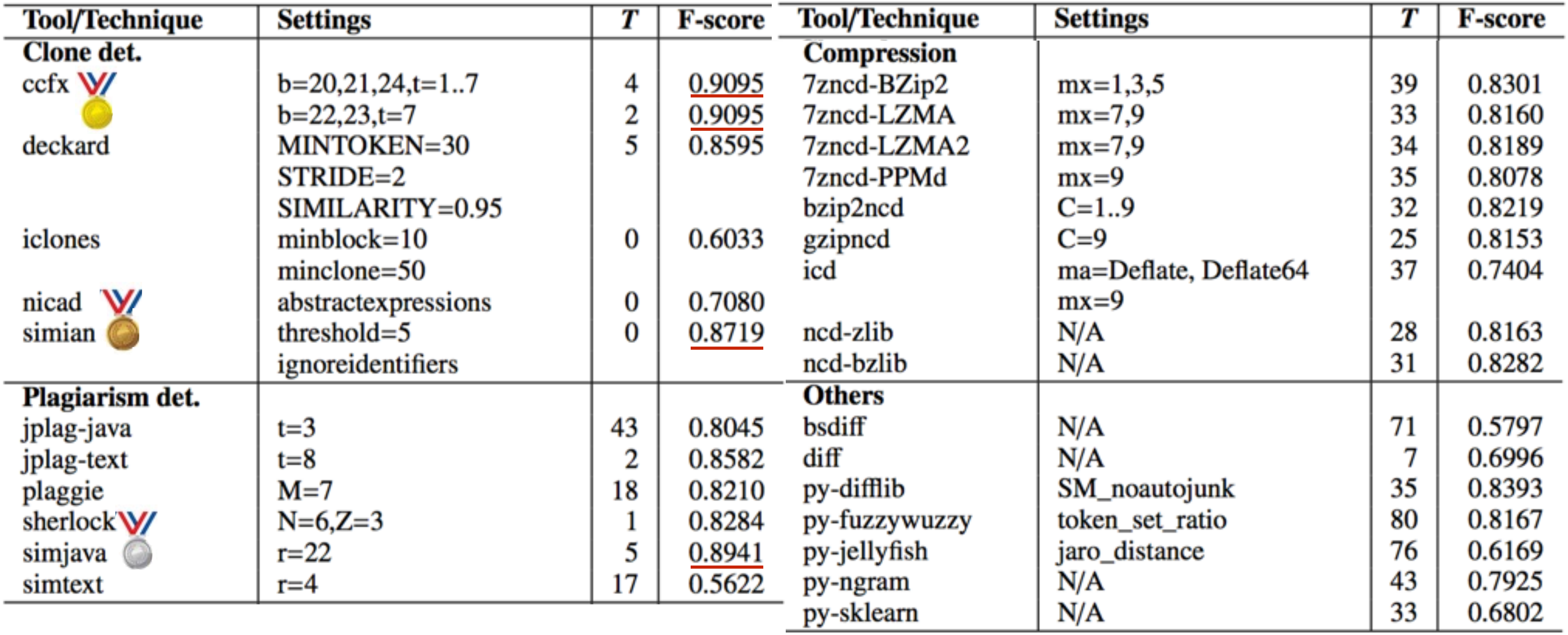
Similarity Report (ncd-bzlib)

	InfC/ orig	InfC/ artfc	InfC/ orig no kraka tau	InfC/ orig no procy on	InfC/ orig pg kraka tau	InfC/ orig pg procy on	InfC/ artfc no kraka tau	InfC/ artfc no procy on	InfC/ artfc pg kraka tau	InfC/ artfc pg procy on	Sqrt/ orig	Sqrt/ artfc	…	Squr/ artfc pg kraka tau	Squr/ artfc pg procy on
InfConv/orig	**100**	55	36	63	32	43	34	60	31	43	20	20	…	14	17
InfConv/artifice	55	**100**	35	54	33	39	37	56	32	39	19	30	…	14	17
InfConv/orig_no_krakatau	36	35	**100**	38	60	26	80	35	59	26	13	14	…	28	17
InfConv/orig_no_procyon	63	54	38	**100**	34	58	37	80	34	58	21	20	…	15	21
InfConv/orig_pg_krakatau	32	33	60	34	**100**	33	61	33	82	33	17	17	…	29	20
InfConv/orig_pg_procyon	43	39	26	58	33	**100**	26	59	33	100	19	20	…	14	21
InfConv/artific_no_krakatau	34	37	80	37	61	26	**100**	36	59	26	14	14	…	28	17
InfConv/artifice_no_procyon	60	56	35	80	33	59	36	**100**	32	59	19	20	…	15	19
InfConv/artifice_pg_krakatau	31	32	59	34	82	33	59	32	**100**	33	15	16	…	28	17
InfConv/artifice_pg_procyon	43	39	26	58	33	100	26	59	33	**100**	19	20	…	14	21
Sqrt/orig	20	19	13	21	17	19	14	19	15	19	**100**	32	…	14	16
Sqrt/artifice	20	30	14	20	17	20	14	20	16	20	32	**100**	…	15	18
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
Square/artifice_pg_krakatau	14	14	28	15	29	14	28	15	28	14	14	15	…	**100**	32
Square/artifice_pg_procyon	17	17	17	21	20	21	17	19	17	21	16	18	…	32	**100**

Optimal Threshold



RQ1: Tool Performance Comparison
RQ2: Best configurations



Tool/Technique	Settings	*T*	F-score
Clone det.			
ccfx 🥇	b=20,21,24,t=1..7	4	0.9095
	b=22,23,t=7	2	0.9095

deckard	MINTOKEN=30
	STRIDE=2
	SIMILARITY=0.95
		5	0.8595

iclones	minblock=10
	minclone=50
		0	0.6033

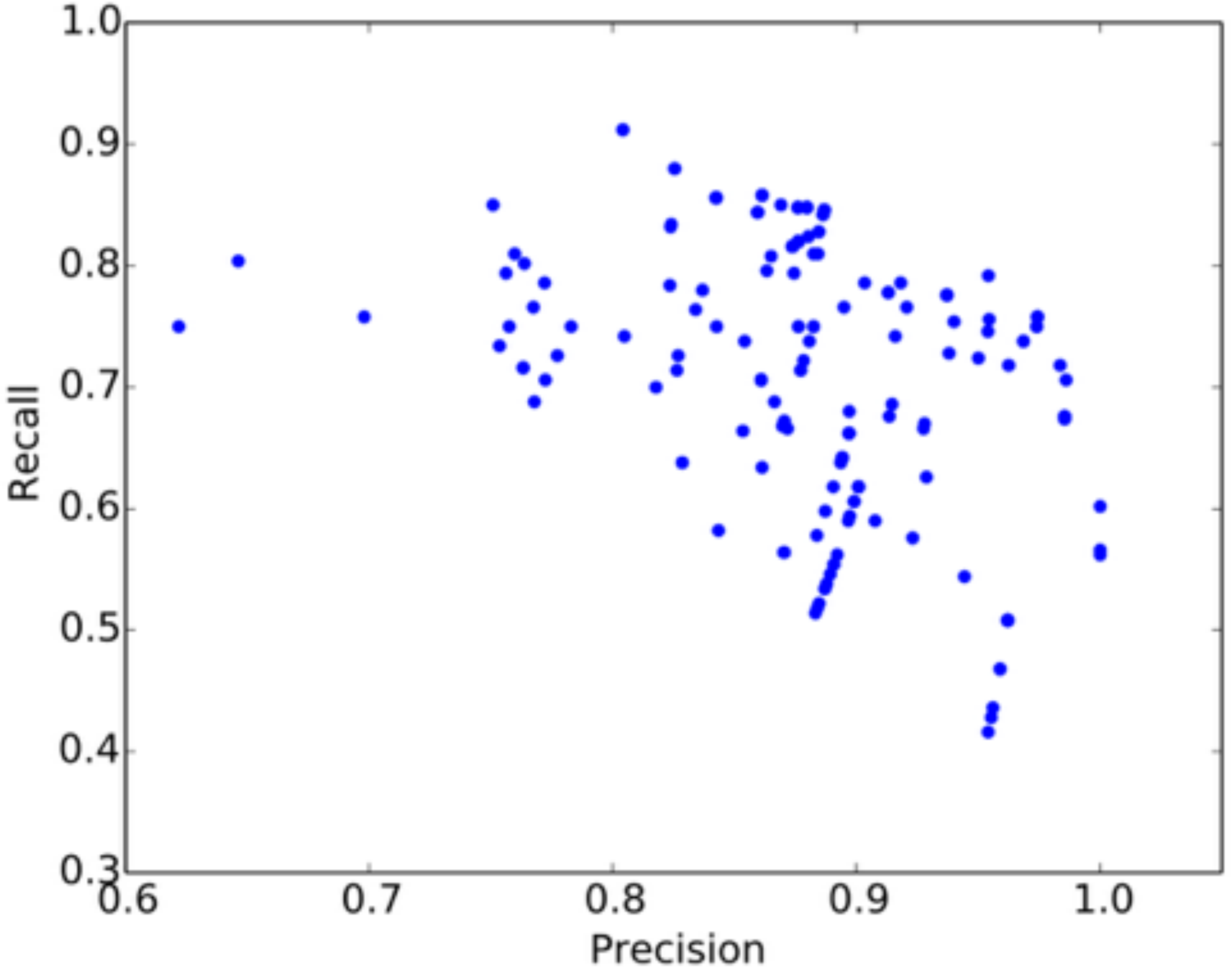
nicad	abstractexpressions	0	0.7080
simian 🥉	threshold=5 ignoreidentifiers	0	0.8719
Plagiarism det.			
jplag-java	t=3	43	0.8045
jplag-text	t=8	2	0.8582
plaggie	M=7	18	0.8210
sherlock	N=6,Z=3	1	0.8284
simjava 🥈	r=22	5	0.8941
simtext	r=4	17	0.5622

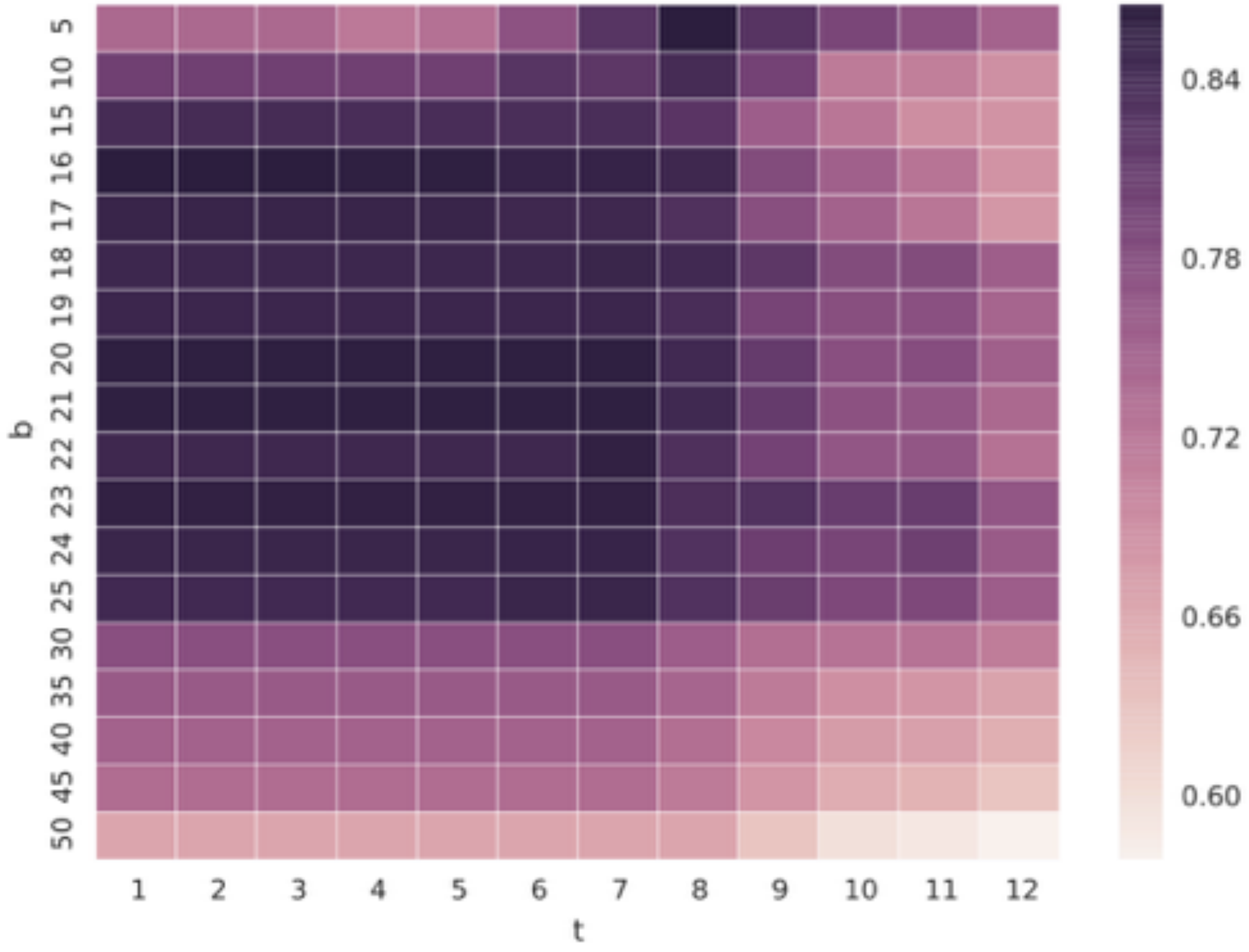
Tool/Technique	Settings	*T*	F-score
Compression			
7zncd-BZip2	mx=1,3,5	39	0.8301
7zncd-LZMA	mx=7,9	33	0.8160
7zncd-LZMA2	mx=7,9	34	0.8189
7zncd-PPMd	mx=9	35	0.8078
bzip2ncd	C=1..9	32	0.8219
gzipncd	C=9	25	0.8153

icd	ma=Deflate, Deflate64
	mx=9
		37	0.7404

ncd-zlib	N/A	28	0.8163
ncd-bzlib	N/A	31	0.8282
Others			
bsdiff	N/A	71	0.5797
diff	N/A	7	0.6996
py-difflib	SM_noautojunk	35	0.8393
py-fuzzywuzzy	token_set_ratio	80	0.8167
py-jellyfish	jaro_distance	76	0.6169
py-ngram	N/A	43	0.7925
py-sklearn	N/A	33	0.6802

CCFinder: Best configurations

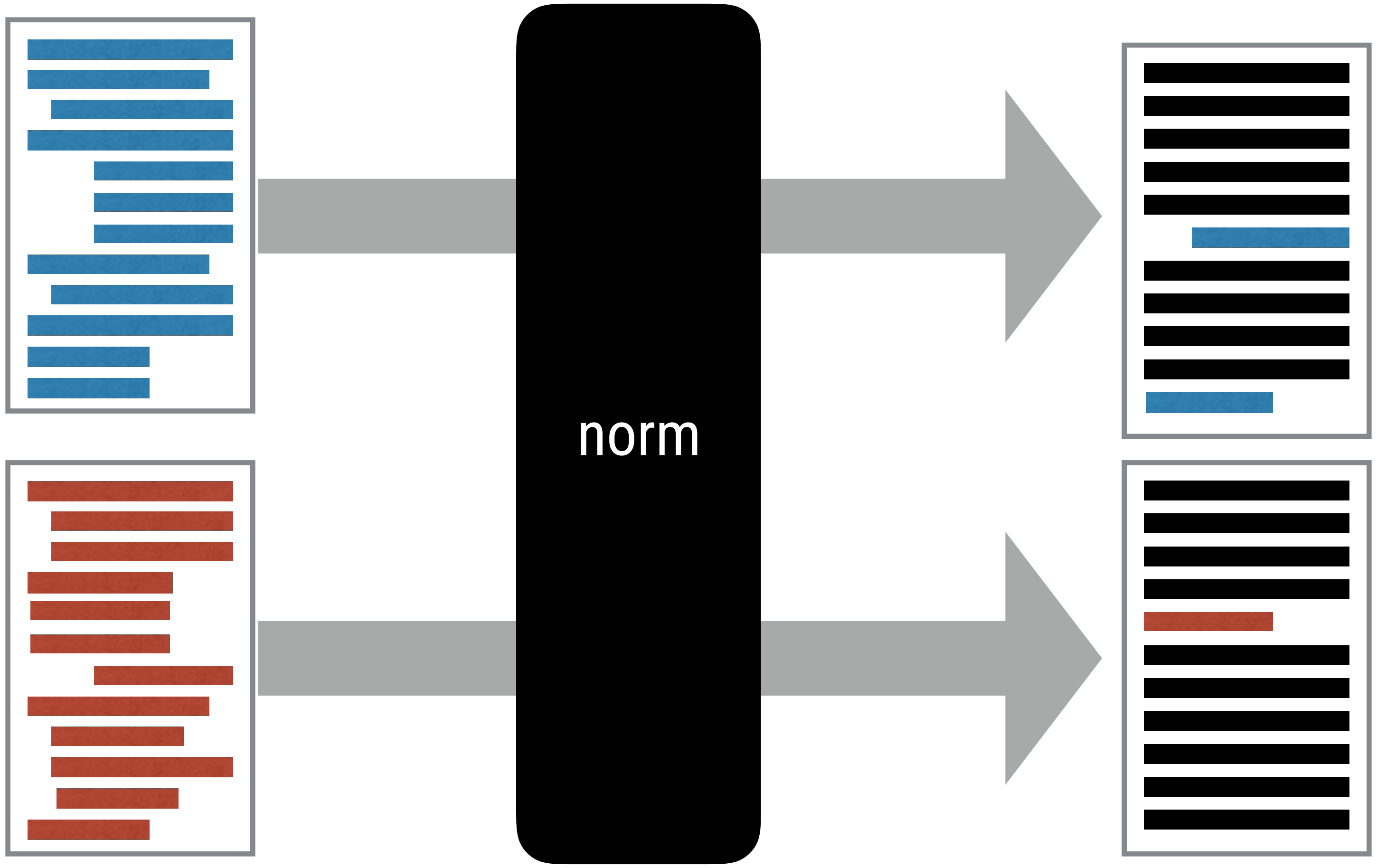




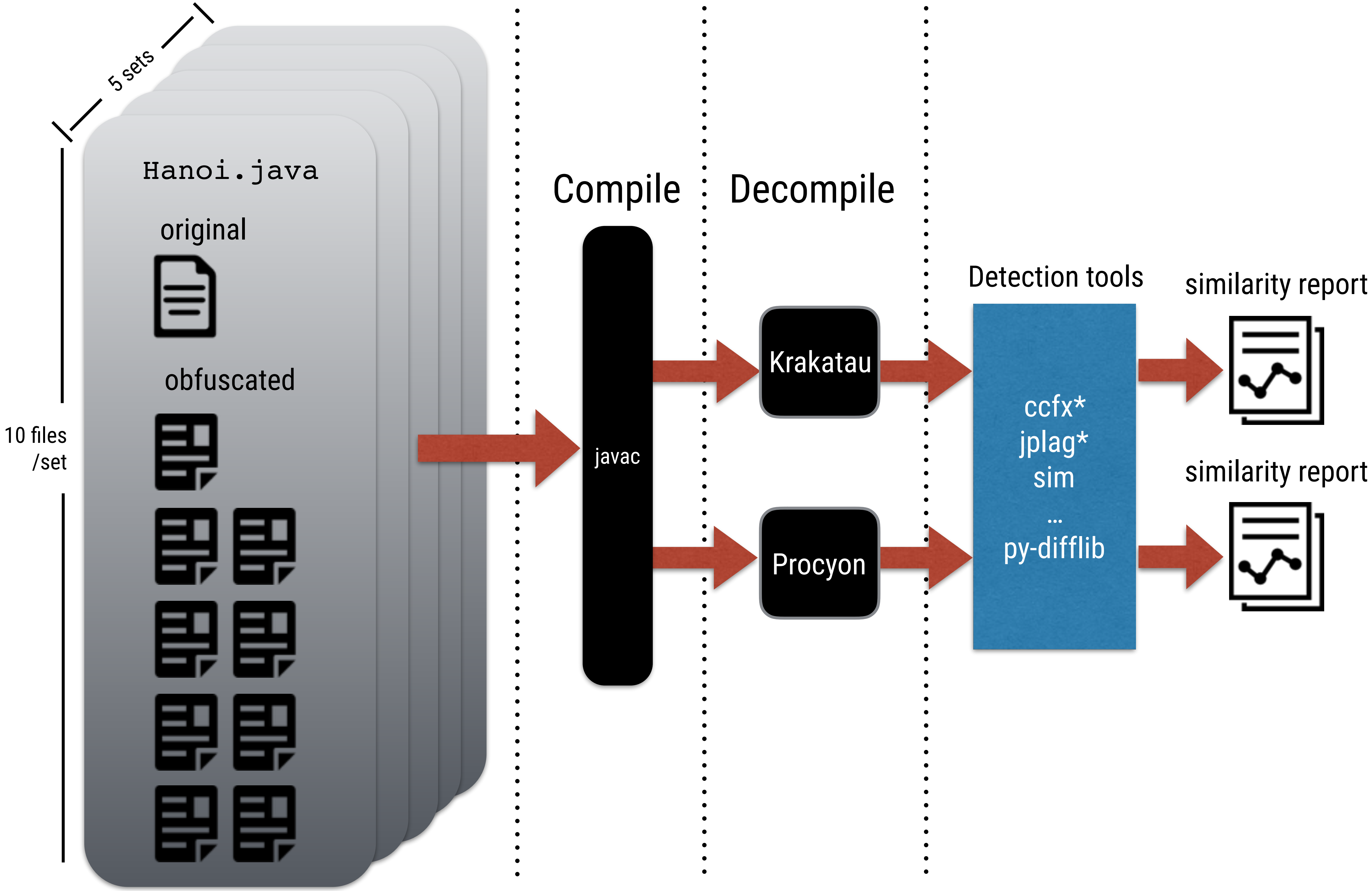
Scenario 2

Decompilation

Normalisation through compilation/decompilation before the similarity detection.



Compiling/Decompiling Process



RQ3: Normalisation by Decompilation

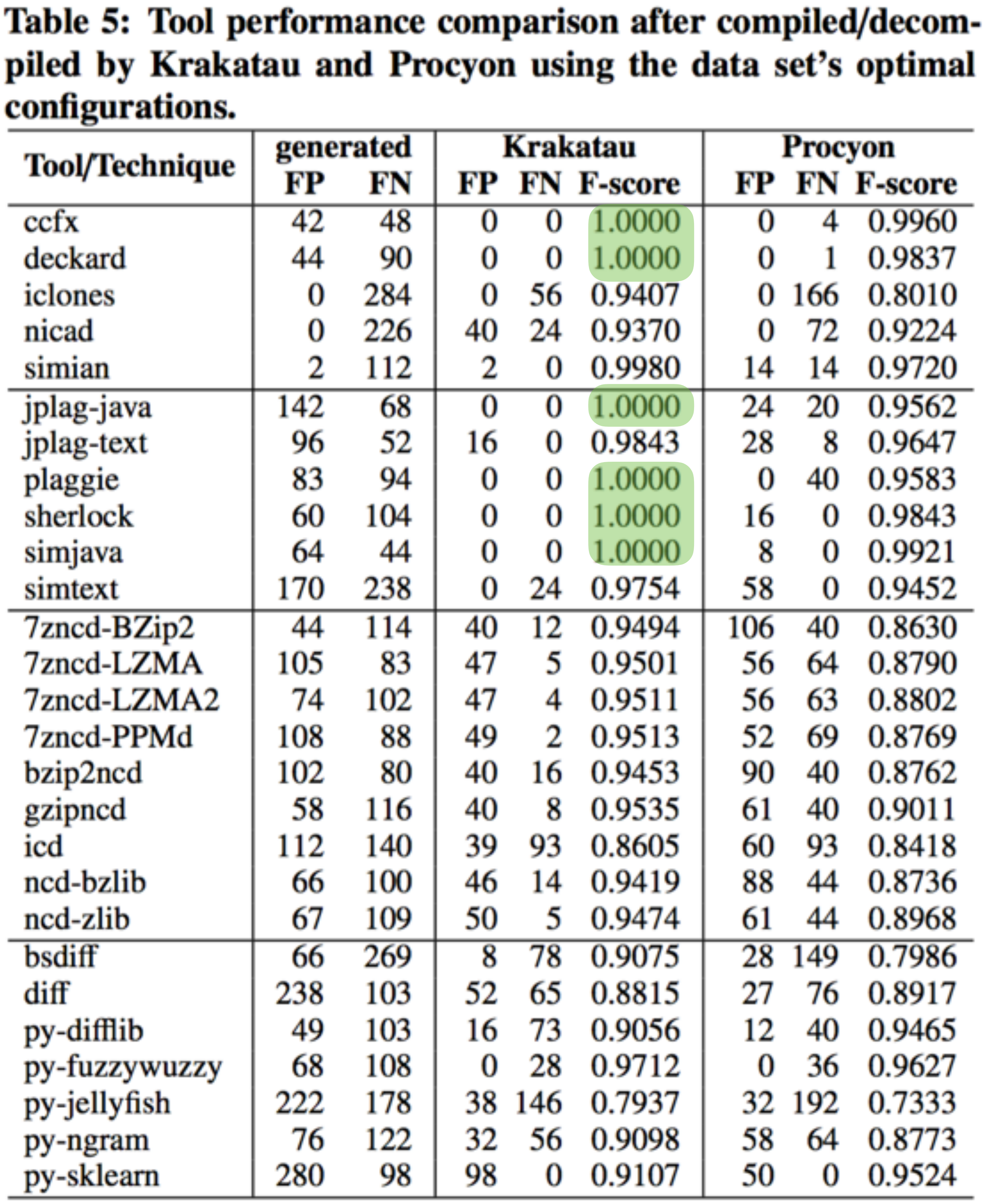


Table 5: Tool performance comparison after compiled/decompiled by Krakatau and Procyon using the data set's optimal configurations.

Tool/Technique	generated		Krakatau			Procyon		
	FP	FN	FP	FN	F-score	FP	FN	F-score
ccfx	42	48	0	0	1.0000	0	4	0.9960
deckard	44	90	0	0	1.0000	0	1	0.9837
iclones	0	284	0	56	0.9407	0	166	0.8010
nicad	0	226	40	24	0.9370	0	72	0.9224
simian	2	112	2	0	0.9980	14	14	0.9720
jplag-java	142	68	0	0	1.0000	24	20	0.9562
jplag-text	96	52	16	0	0.9843	28	8	0.9647
plaggie	83	94	0	0	1.0000	0	40	0.9583
sherlock	60	104	0	0	1.0000	16	0	0.9843
simjava	64	44	0	0	1.0000	8	0	0.9921
simtext	170	238	0	24	0.9754	58	0	0.9452
7zncd-BZip2	44	114	40	12	0.9494	106	40	0.8630
7zncd-LZMA	105	83	47	5	0.9501	56	64	0.8790
7zncd-LZMA2	74	102	47	4	0.9511	56	63	0.8802
7zncd-PPMd	108	88	49	2	0.9513	52	69	0.8769
bzip2ncd	102	80	40	16	0.9453	90	40	0.8762
gzipncd	58	116	40	8	0.9535	61	40	0.9011
icd	112	140	39	93	0.8605	60	93	0.8418
ncd-bzlib	66	100	46	14	0.9419	88	44	0.8736
ncd-zlib	67	109	50	5	0.9474	61	44	0.8968
bsdiff	66	269	8	78	0.9075	28	149	0.7986
diff	238	103	52	65	0.8815	27	76	0.8917
py-difflib	49	103	16	73	0.9056	12	40	0.9465
py-fuzzywuzzy	68	108	0	28	0.9712	0	36	0.9627
py-jellyfish	222	178	38	146	0.7937	32	192	0.7333
py-ngram	76	122	32	56	0.9098	58	64	0.8773
py-sklearn	280	98	98	0	0.9107	50	0	0.9524

Scenario 3 & 4

Semantically Similar Code

Simions: a data set of semantically identical but independently developed Java files

Functions for email address validation – one file contains one implementation)

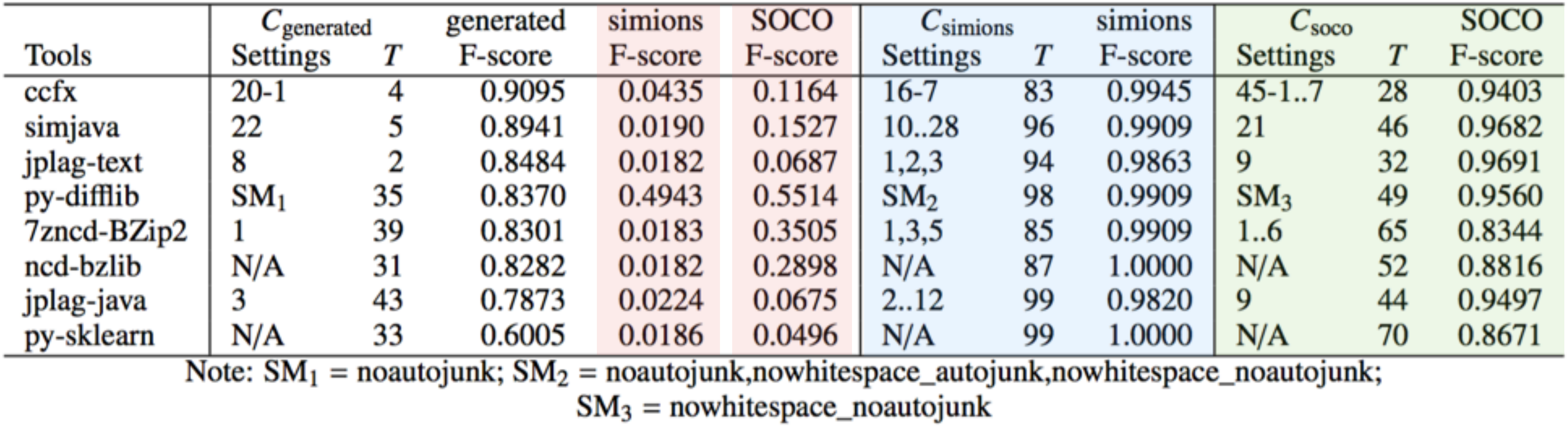
Reused Boiler-plate Code

SOCO (SOurce COde re-use) data set

A competition for discovering monolingual re-used source code amongst a given set of programs.

* E. Juergens, F. Deissenboeck, and B. Hummel. Code similarities beyond copy & paste. In *CSMR'11*, 2011.

RQ4: Reuse of Configurations



	$C_{\text{generated}}$		generated	simions	SOCO	C_{simions}		simions	C_{soco}		SOCO
Tools	Settings	T	F-score	F-score	F-score	Settings	T	F-score	Settings	T	F-score
ccfx	20-1	4	0.9095	0.0435	0.1164	16-7	83	0.9945	45-1..7	28	0.9403
simjava	22	5	0.8941	0.0190	0.1527	10..28	96	0.9909	21	46	0.9682
jplag-text	8	2	0.8484	0.0182	0.0687	1,2,3	94	0.9863	9	32	0.9691
py-difflib	SM_1	35	0.8370	0.4943	0.5514	SM_2	98	0.9909	SM_3	49	0.9560
7zncd-BZip2	1	39	0.8301	0.0183	0.3505	1,3,5	85	0.9909	1..6	65	0.8344
ncd-bzlib	N/A	31	0.8282	0.0182	0.2898	N/A	87	1.0000	N/A	52	0.8816
jplag-java	3	43	0.7873	0.0224	0.0675	2..12	99	0.9820	9	44	0.9497
py-sklearn	N/A	33	0.6005	0.0186	0.0496	N/A	99	1.0000	N/A	70	0.8671

Note: SM_1 = noautojunk; SM_2 = noautojunk,nowhitespace_autojunk,nowhitespace_noautojunk;
SM_3 = nowhitespace_noautojunk

Conclusion

Tools perform differently given the optimised configuration.
Winner: CCFinderX

Compilation/decompilation is effective and can be adopted as a normalisation technique.

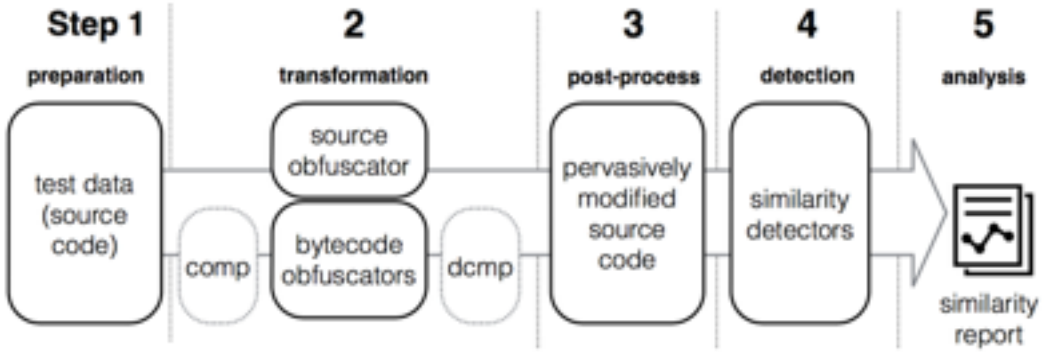
Every technique and tool is extremely sensitive to its own configurations

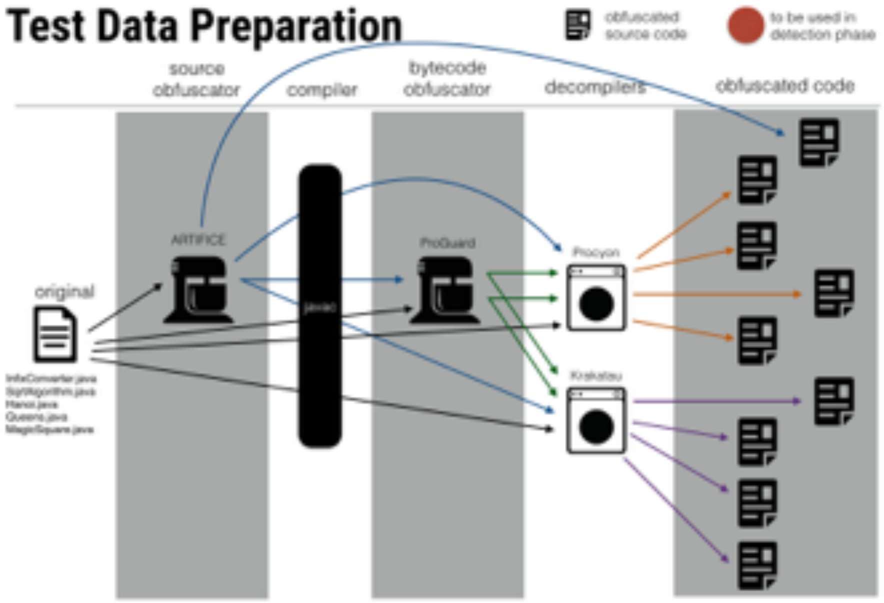
DON'T: reuse default configurations.

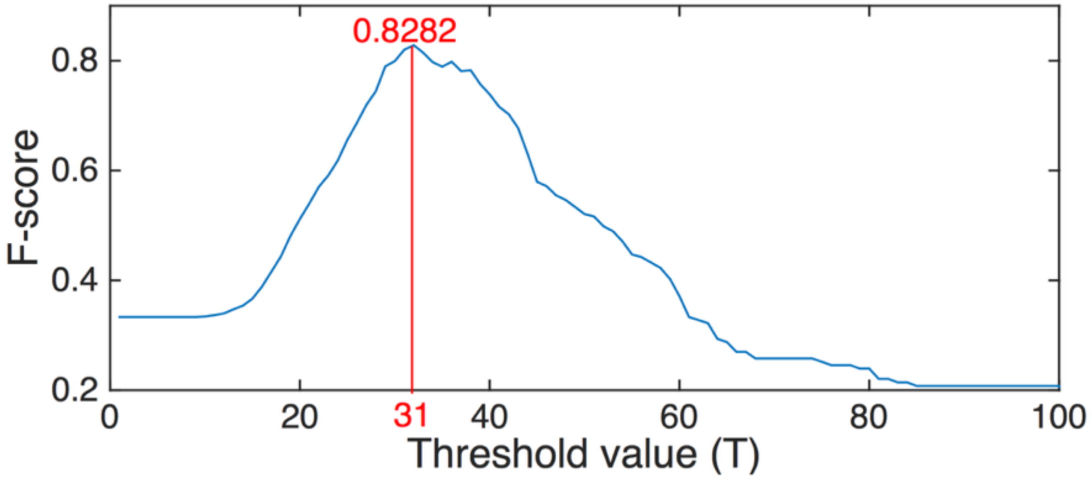
DON'T: reuse of optimal configuration since it is naturally biased by the particular data set.

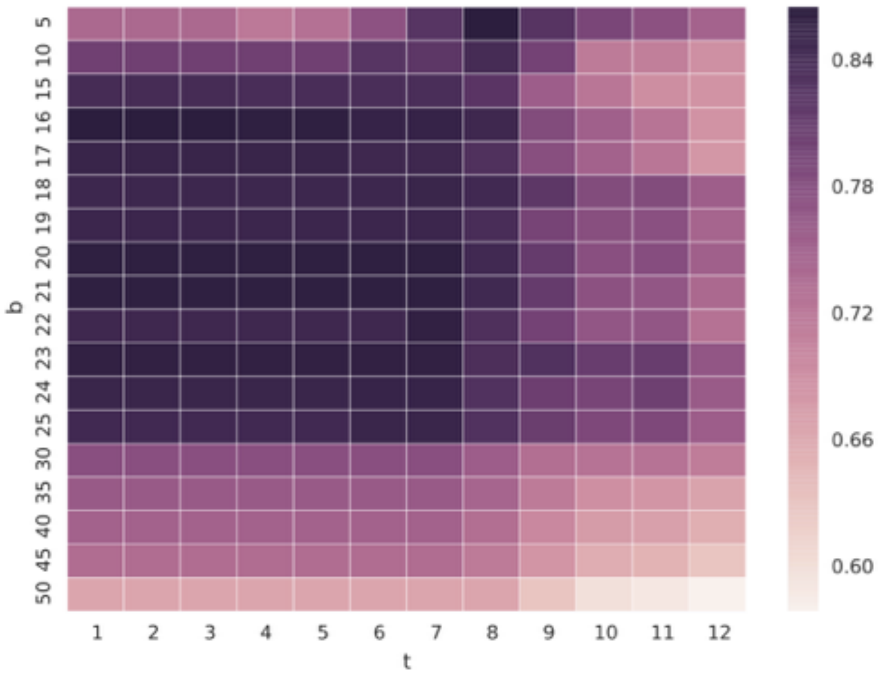
DO: Researchers have to consider this limitation every time when they use similarity detection techniques in their studies.

Similarity of Source Code in the Presence of Pervasive Modifications


Step 1
preparation
test data (source code)
2
transformation
source obfuscator
comp
bytecode obfuscators
dcmp
3
post-process
pervasively modified source code
4
detection
similarity detectors
5
analysis
similarity report


Test Data Preparation
obfuscated source code
to be used in detection phase
source obfuscator
compiler
bytecode obfuscator
decompilers
obfuscated code
original
InfoConverter.java
SqrtAlgorithm.java
Hanoi.java
Queens.java
MagicSquare.java
ARTIFICE
javac
ProGuard
Procyon
Krakatau


F-score
Threshold value (T)


b
5
10
15
16
17
18
19
20
21
22
23
24
25
30
35
40
45
50
t
1
2
3
4
5
6
7
8
9
10
11
12
0.84
0.78
0.72
0.66
0.60