A Comparison of Code Similarity Analyzers

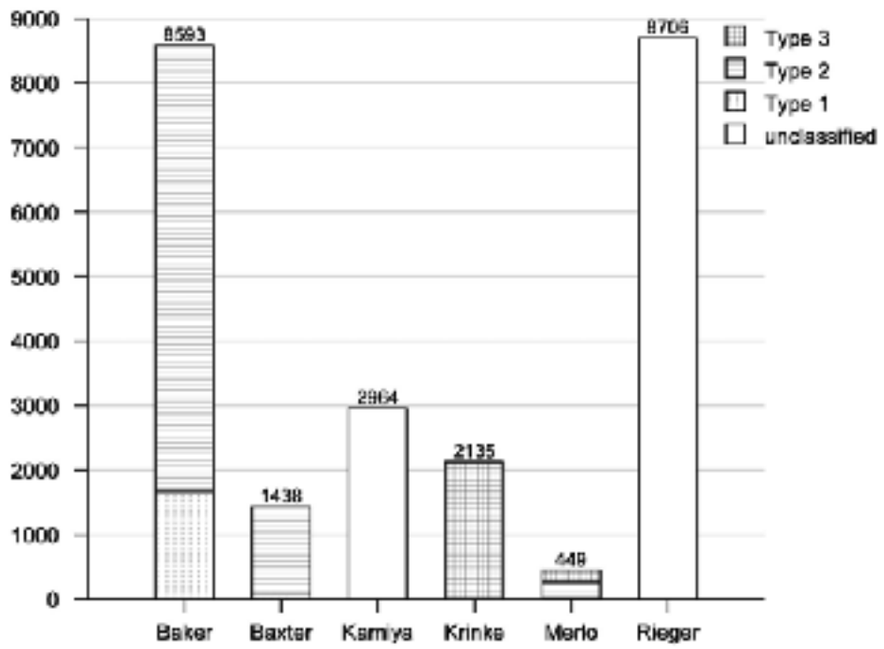
C. Ragkhitwetsagul, J. Krinke, D. Clark

SCAM ’16, EMSE (under reviewed)

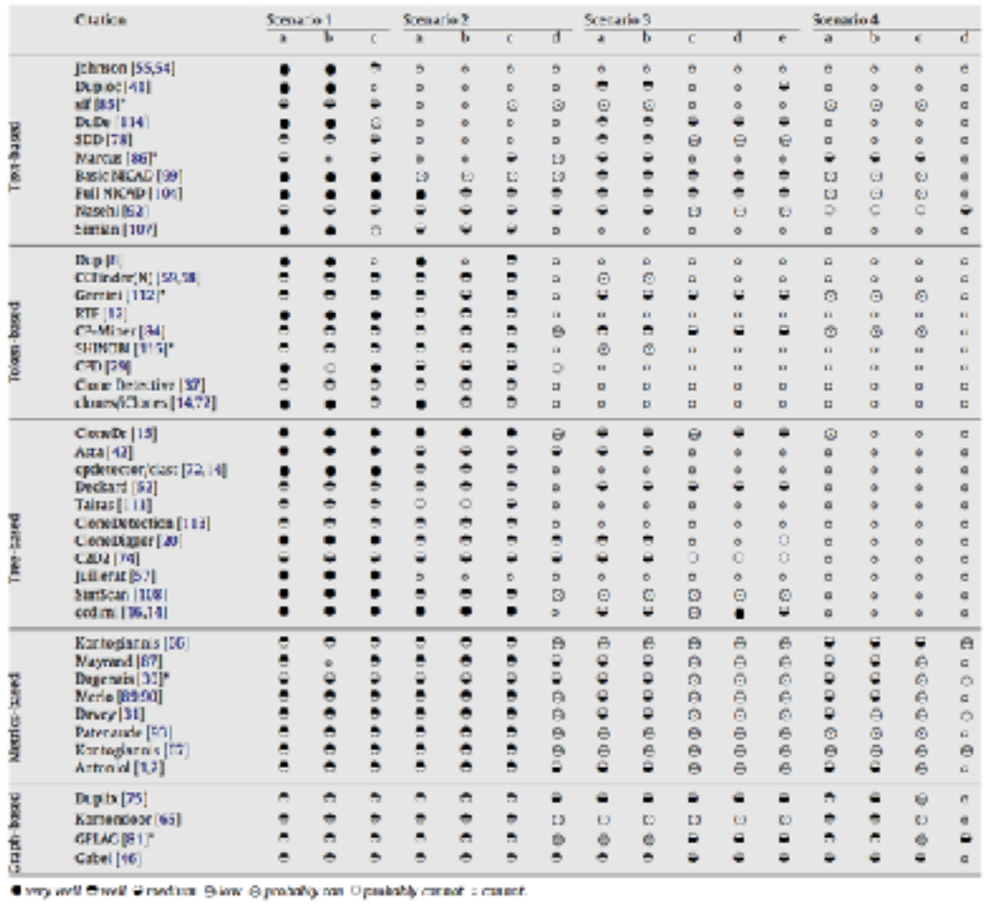
Photo: https://c1.staticflickr.com/1/316/31831180223_38db905f28_c.jpg

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

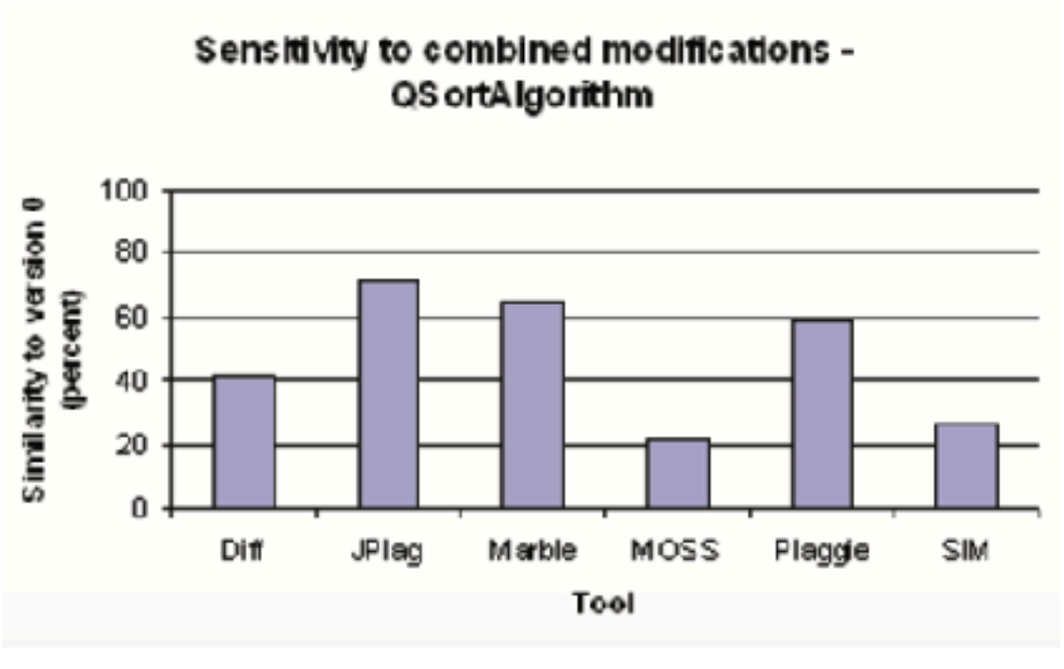
Bellon et al. (TSE 2007)



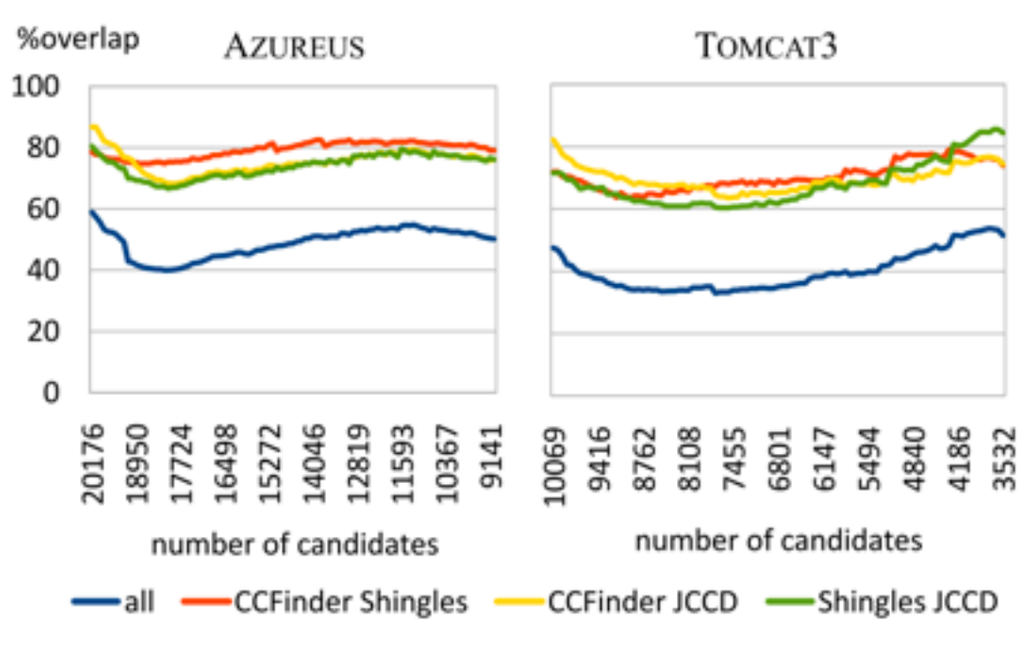
Roy et al. (Sci Comp Prog. 2009)



Hage et al. (CSERC 2010)



Biegel et al. (MSR '11)



1. The selected tools are limited to only a subset of clone or plagiarism detectors (and their parameters).

2. The results are based on different data sets.

Table 2: Tools with their similarity measures

Pervasive Modifications

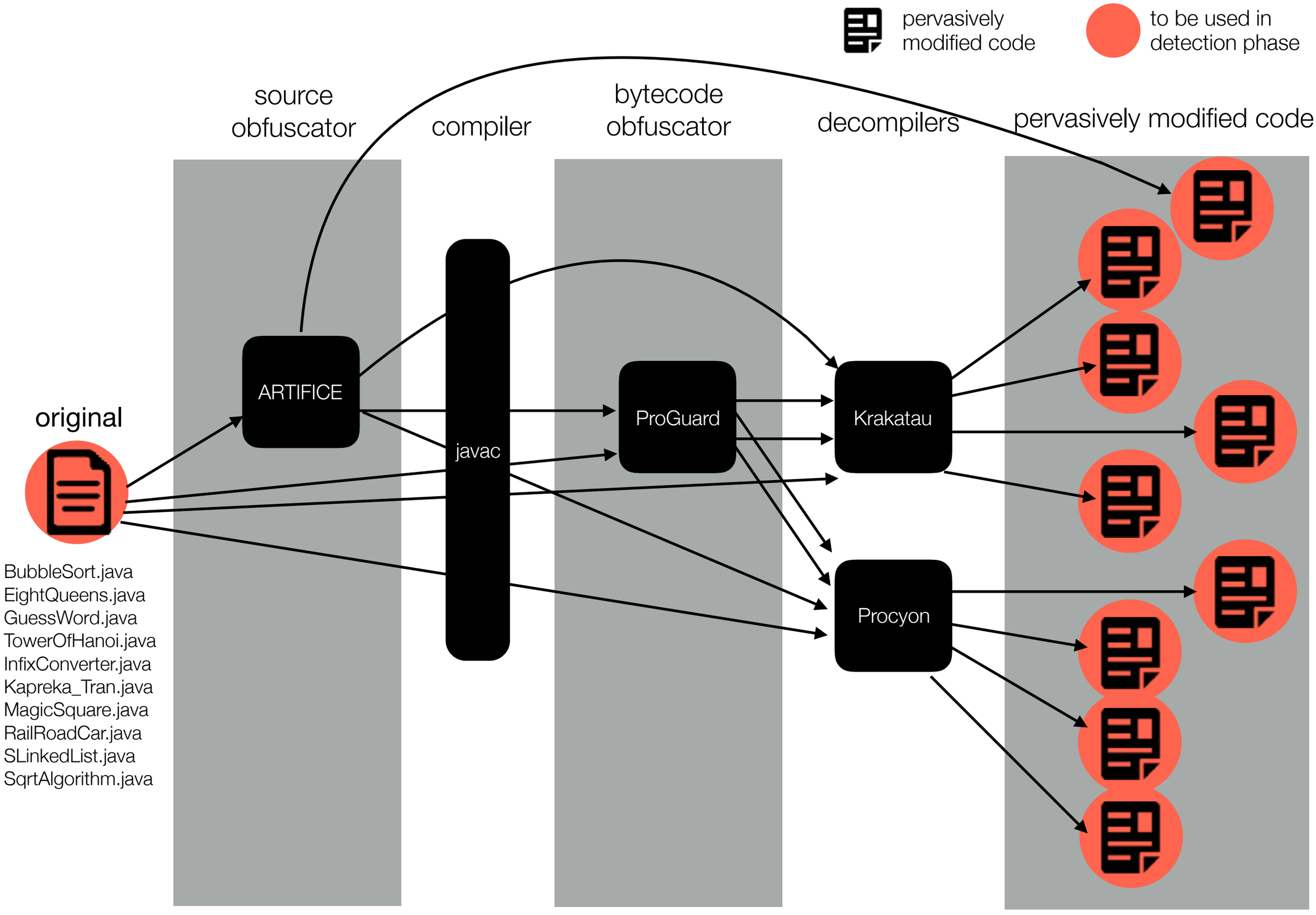
From: https://www.princeton.edu/pr/pub/integrity/pages/plagiarism/

```java
/* ORIGINAL */
private static int partition
     (Comparable[] a, int lo, int hi) {
    int i = lo;
    int j = hi+1;
    Comparable v = a[lo];
    while (true) {
        while (less(a[++i], v)) {
            if (i == hi) break;
        }
        while (less(v, a[--j])) {
            if (j == lo) break;
        }
        if (i >= j) break;
        exch(a, i, j);
    }
    exch(a, lo, j);
    return j;
}
```

```java
/* PERVASIVELY MODIFIED CODE */
private static int partition
     (int[] bob, int left, int right){
    int x = left;
    int y = right+1;
    for (;;) {
        while (less(bob[left],bob[--y]))
            if (y == left) break;
        while (less(bob[++x],bob[left]))
            if (x == right) break;
        if (x >= y) break;
        swap(bob, y, x);
    }
    swap(bob, y, left);
    return y;
}
```

SW Plagiarism | clone evolution | refactoring

Code modifications	Artifice	ProGuard	(De)compilers
Lexical modification			
Formatting changes (Roy and Cordy, 2009; Duric and Gasevic, 2013; Joy and Luck, 1999)	✓		✓
Addition, modification or deletion of comments (Duric and Gasevic, 2013; Joy and Luck, 1999)	✓		✓
Renaming of identifiers, methods (Roy and Cordy, 2009; Duric and Gasevic, 2013; Joy and Luck, 1999; Brixtel et al, 2010; Fowler, 2013)	✓	✓	✓
Modification of constant values (Duric and Gasevic, 2013)		✓	
Structural modification			
Split or merge of variable declarations (Duric and Gasevic, 2013)	✓		✓
Addition, modification or deletion of modifiers (Duric and Gasevic, 2013; Fowler, 2013)		✓	✓
Line insertion/deletion with further edits (Roy and Cordy, 2009)	✓		✓
Reordering of statements & control replacements (Roy and Cordy, 2009; Duric and Gasevic, 2013; Joy and Luck, 1999; Brixtel et al, 2010)	✓	✓	✓
Modification of control structures (Duric and Gasevic, 2013; Joy and Luck, 1999; Brixtel et al, 2010)	✓		✓
Changing of data types and modification of data structures (Duric and Gasevic, 2013)			✓
Method inlining and method refactoring (Duric and Gasevic, 2013; Fowler, 2013)		✓	
Structural redesign of source code (Duric and Gasevic, 2013; Fowler, 2013)			✓


pervasively modified code
to be used in detection phase
source obfuscator
compiler
bytecode obfuscator
decompilers
pervasively modified code
original
ARTIFICE
javac
ProGuard
Krakatau
Procyon
BubbleSort.java
EightQueens.java
GuessWord.java
TowerOfHanoi.java
InfixConverter.java
Kapreka_Tran.java
MagicSquare.java
RailRoadCar.java
SLinkedList.java
SqrtAlgorithm.java

Boiler-Plate Code

153.java (active)

```
171 }
172         byte b3;
173         if ( i >= bytes.length ) {
174             b2 = 0;
175             b3 = 0;
176             pad = 2;
177         } else {
178             b2 = bytes [i++];
179             if ( i >= bytes.length ) {
180                 b3 = 0;
181                 pad = 1;
182             } else {
183                 b3 = bytes [i++];
184             }
185         }
186         byte c1 = ( byte ) ( b1 >> 2 );
187         byte c2 = ( byte ) ( ( ( b1 & 0x3 ) << 4 ) |
188         byte
189         enco
190         enco
191         swit
192         case
193
194             encodedString += base64Array [c4];
195             break;
196         case 1:
197             encodedString += base64Array [c3];
198             encodedString += "=";
199             break;
200         case 2:
201             encodedString += "==";
202             break;
203         }
204     }
205     return "" + encodedString;
206
207 }
208 }
```

086.java (other)

```
171         byte b3;
172         if ( i >= bytes.length ) {
173             b2 = 0;
174             b3 = 0;
175             pad = 2;
176         } else {
177             b2 = bytes [i++];
178             if ( i >= bytes.length ) {
179                 b3 = 0;
180                 pad = 1;
181             } else {
182                 b3 = bytes [i++];
183             }
184         }
185         byte c1 = ( byte ) ( b1 >> 2 );
186         byte c2 = ( byte ) ( ( ( b1 & 0x3 ) << 4 ) |
187                                          0xf ) << 2 ) |
188                                              f );
189                                             [c1];
190                                 ing += base64Array [c2];
191                                 ad ) {
192
193                                 dedString += base64Array [c3];
194             encodedString += base64Array [c4];
195             break;
196         case 1:
197             encodedString += base64Array [c3];
198             encodedString += "=";
199             break;
200         case 2:
201             encodedString += "==";
202             break;
203         }
204     }
205
206     return encodedString;
207 }
208 }
```

Flores E., Rosso P., Moreno L., Villatoro-Tello E. (2014)
Detection of SOurce COde re-use (SOCO).
http://users.dsic.upv.es/grupos/nle/soco/

Parameter Settings

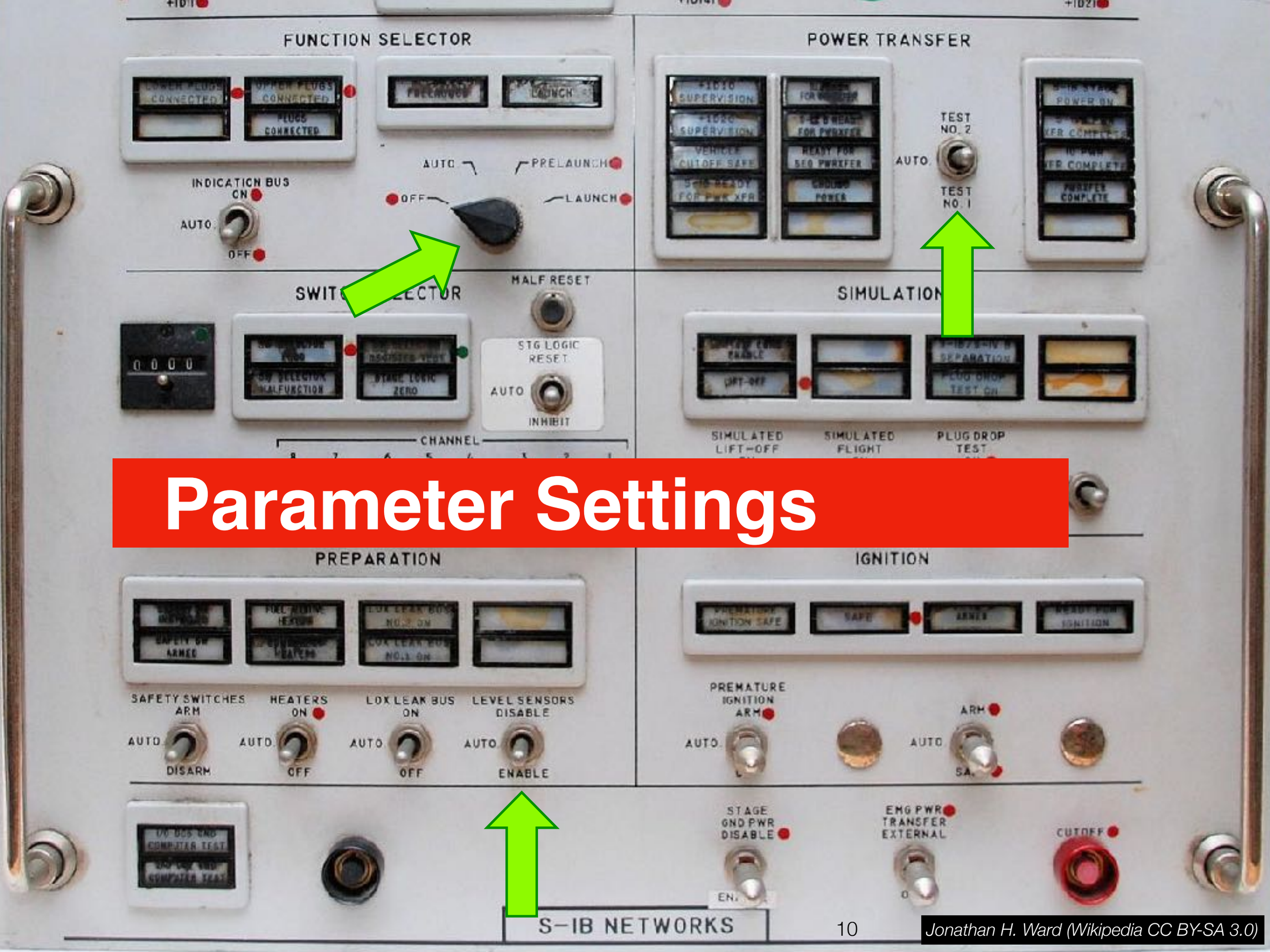


Table 3: Tools and their parameters with chosen value ranges (DF denotes default parameters)

Tool	Settings	Details	DF	Range
Clone det.				
ccfx	b	min no. of tokens	50	3 4 5 10 15 16 17 18 19 20 21 22 23 24 25 30 35 40 45 50
	t	min token kinds	12	1 2 3 .. 14
deckard	mintoken	min no. of tokens	50	30, 50
	stride	sliding window size	inf	0, 1, 2, inf
	similarity	clone similarity	1.0	0.90, 0.95, 1.00
iclones	minblock	min token length	20	8 10 20 30 40 50
	minclone	min no. of tokens	100	50 60 .. 140 150
nicad	UPI	% of unique code	0.30	0.30, 0.50
	minline	min no. of lines	10	5, 8, 10
	rename	variable renaming	none	blind, consistent
	abstract	code abstraction	none	none, declaration, statement, expression, condition, literal
simian	threshold	min no. of lines	6	3 4 5 .. 10
	options	other options	none	none, ignoreCharacters, ignoreIdentifiers, ignoreLiterals, ignoreVariableNames
Plagiarism det.				
jplag-java	t	min no. of tokens	9	1 2 3 .. 12
jplag-text	t	min no. of tokens	9	1 2 3 .. 12
plaggie	M	min no. of tokens	11	1 2 3 .. 14

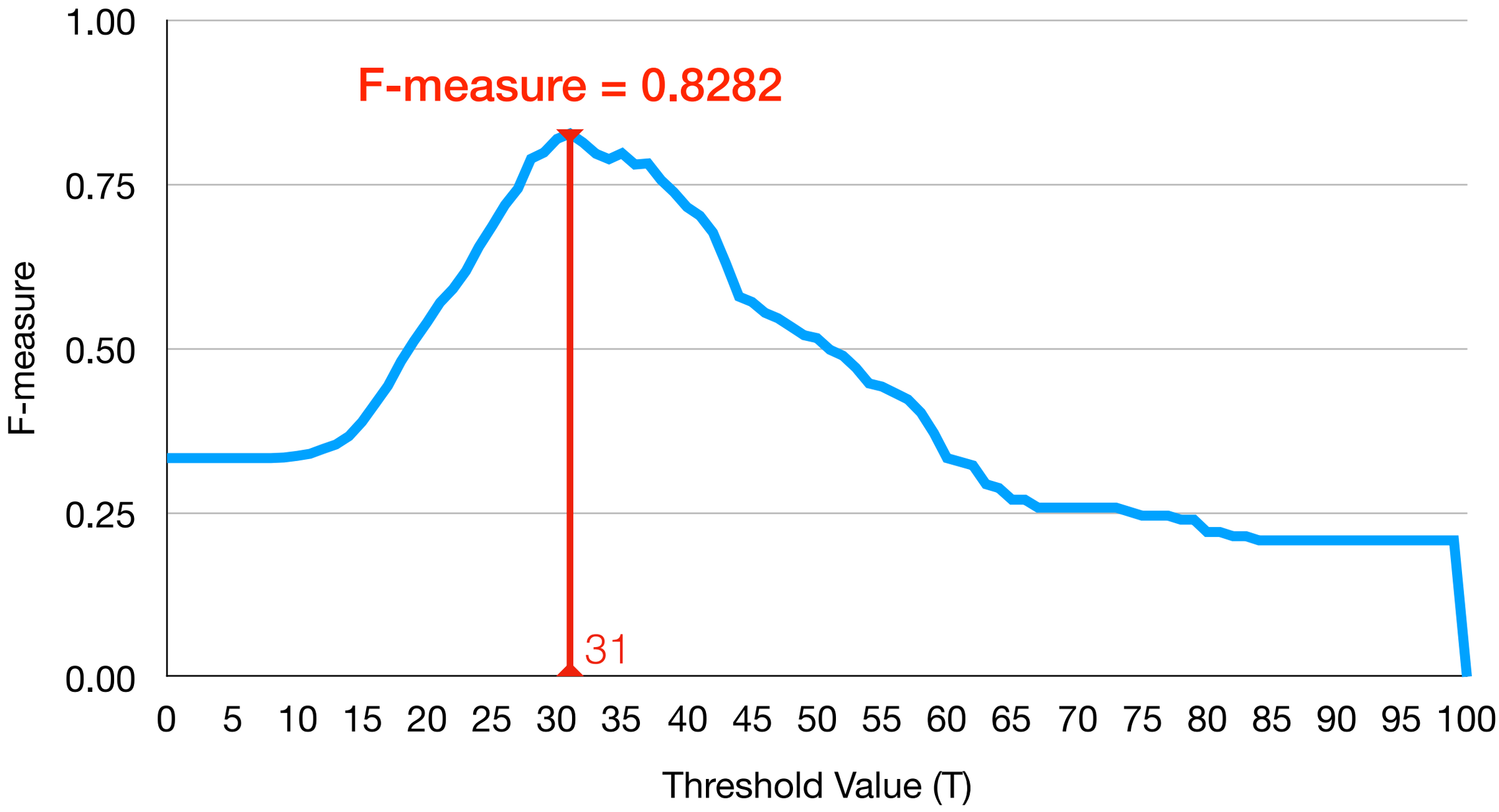
Similarity Report

	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**

Similarity Threshold = 50

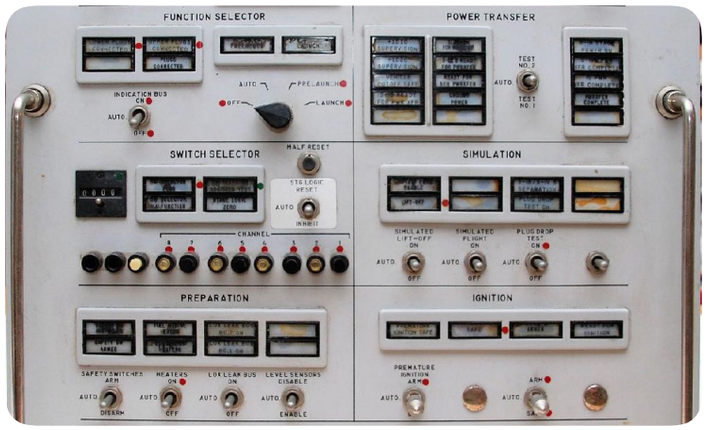
	InfC/ orig	InfC/ artfc	InfC/ orig no kraka tau	InfC/ orig no procy on	InfC/ orig pg kraka tau	InfC/ orig pg procy on	InfC/ artfc no kraka tau	InfC/ artfc no procy on	InfC/ artfc pg kraka tau	InfC/ artfc pg procy on	Sqrt/ orig	Sqrt/ artfc	…	Squr/ artfc pg kraka tau	Squr/ artfc pg procy on
InfConv/orig	**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**

Best Threshold

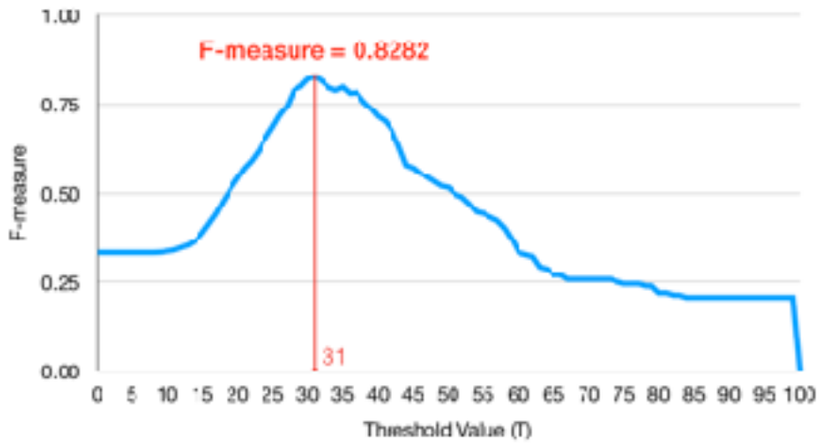


Optimal Configuration

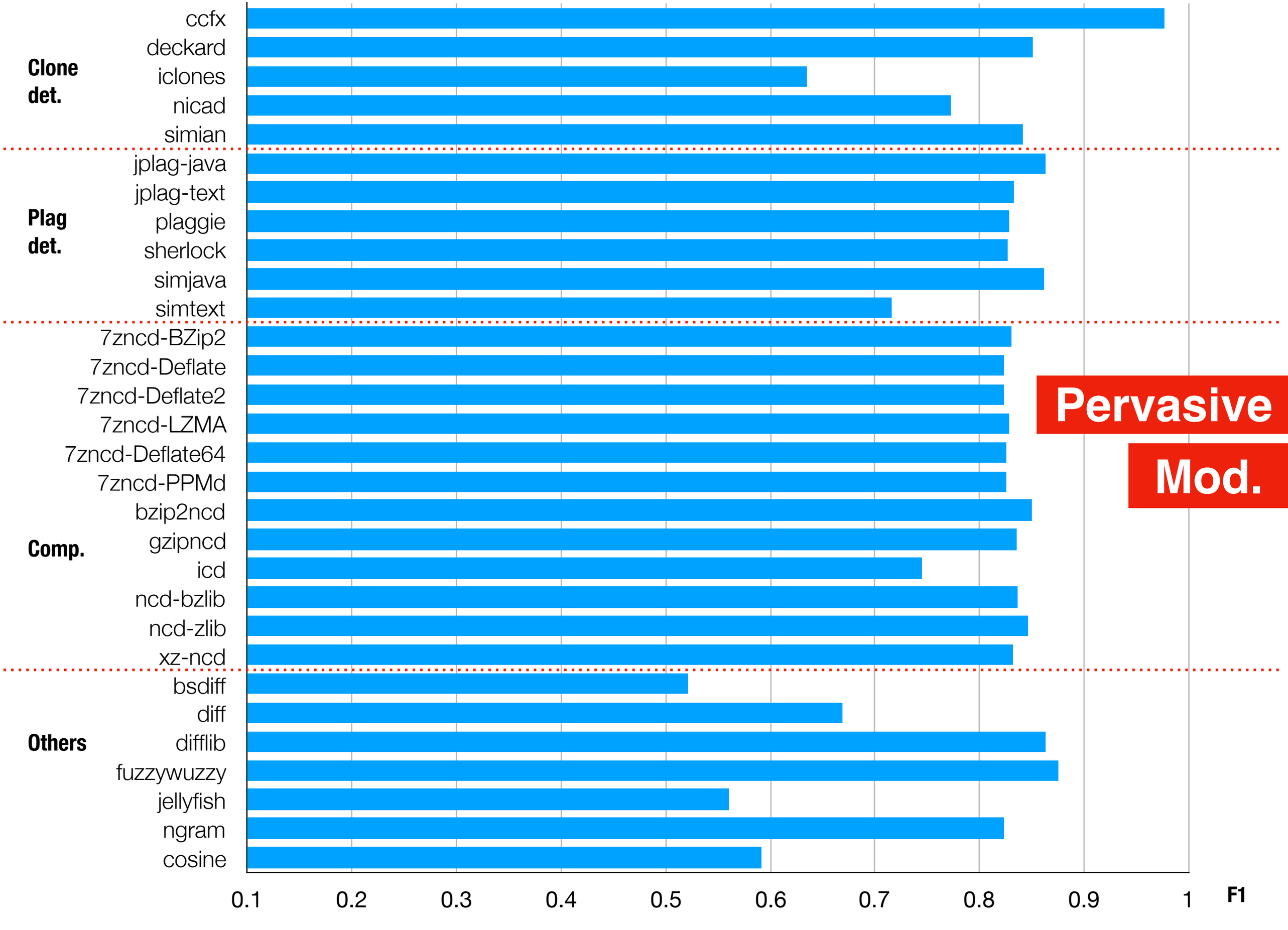
Best Param Settings



Best Threshold



Pervasive: 14,880,000 pairwise comparisons
SOCO: 99,816,528 pairwise comparisons


Clone det.
ccfx
deckard
iclones
nicad
simian
Plag det.
jplag-java
jplag-text
plaggie
sherlock
simjava
simtext
Comp.
7zncd-BZip2
7zncd-Deflate
7zncd-Deflate2
7zncd-LZMA
7zncd-Deflate64
7zncd-PPMd
bzip2ncd
gzipncd
icd
ncd-bzlib
ncd-zlib
xz-ncd
Others
bsdiff
diff
difflib
fuzzywuzzy
jellyfish
ngram
cosine
Pervasive
Mod.
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
F1

Clone det.

ccfx
deckard
iclones
nicad
simian

Plag det.

jplag-java
jplag-text
plaggie
sherlock
simjava
simtext

Comp.

7zncd-BZip2
7zncd-Deflate
7zncd-Deflate2
7zncd-LZMA
7zncd-Deflate64
7zncd-PPMd
bzip2ncd
gzipncd
icd
ncd-bzlib
ncd-zlib
xz-ncd

Others

bsdiff
diff
difflib
fuzzywuzzy
jellyfish
ngram
cosine

Boiler-Plate

0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

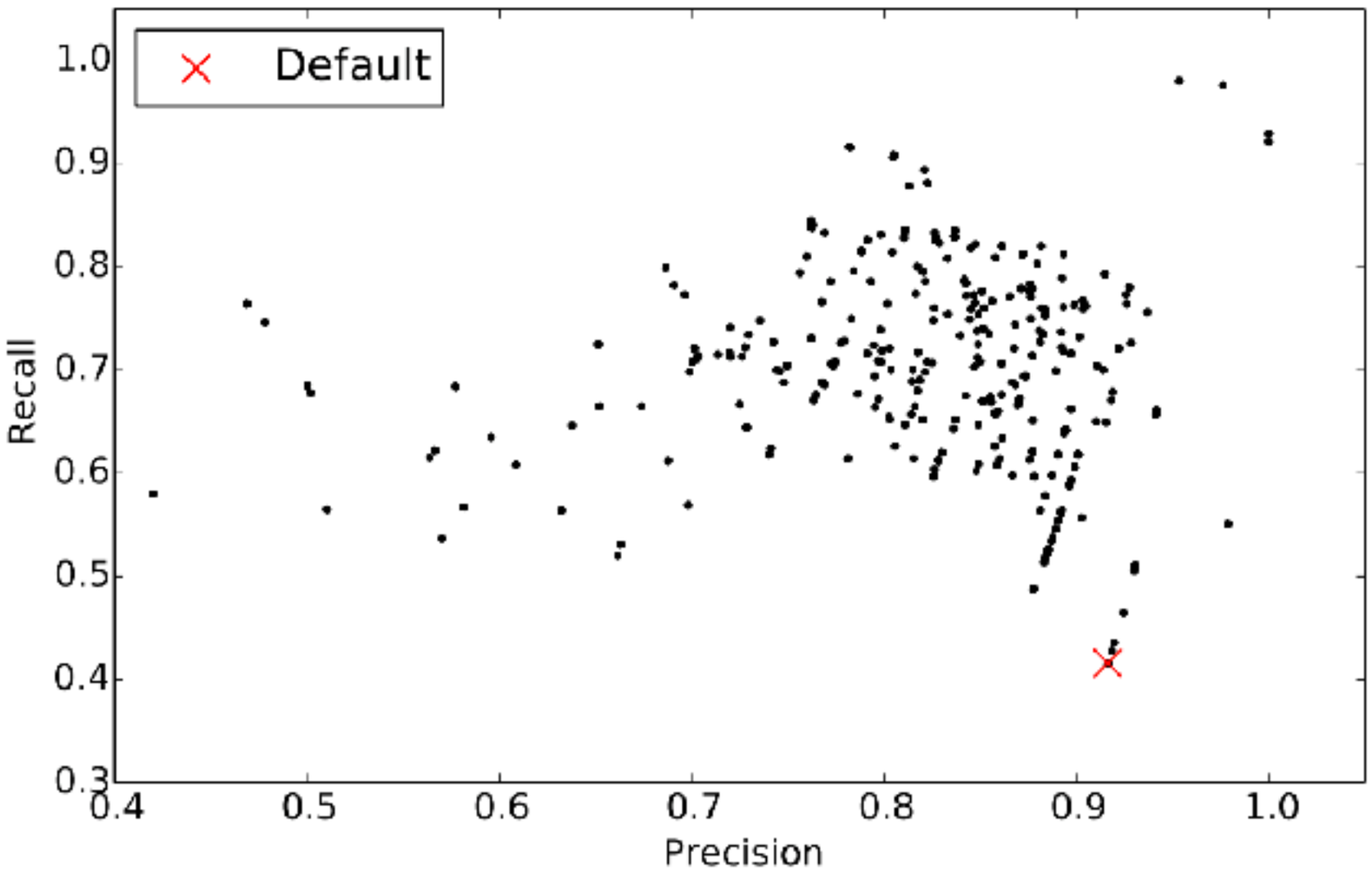
F1

Highly specialised source code similarity detection techniques and tools can perform better than more general, compression & textual similarity measures.

Interesting: difflib and fuzzywuzzy.

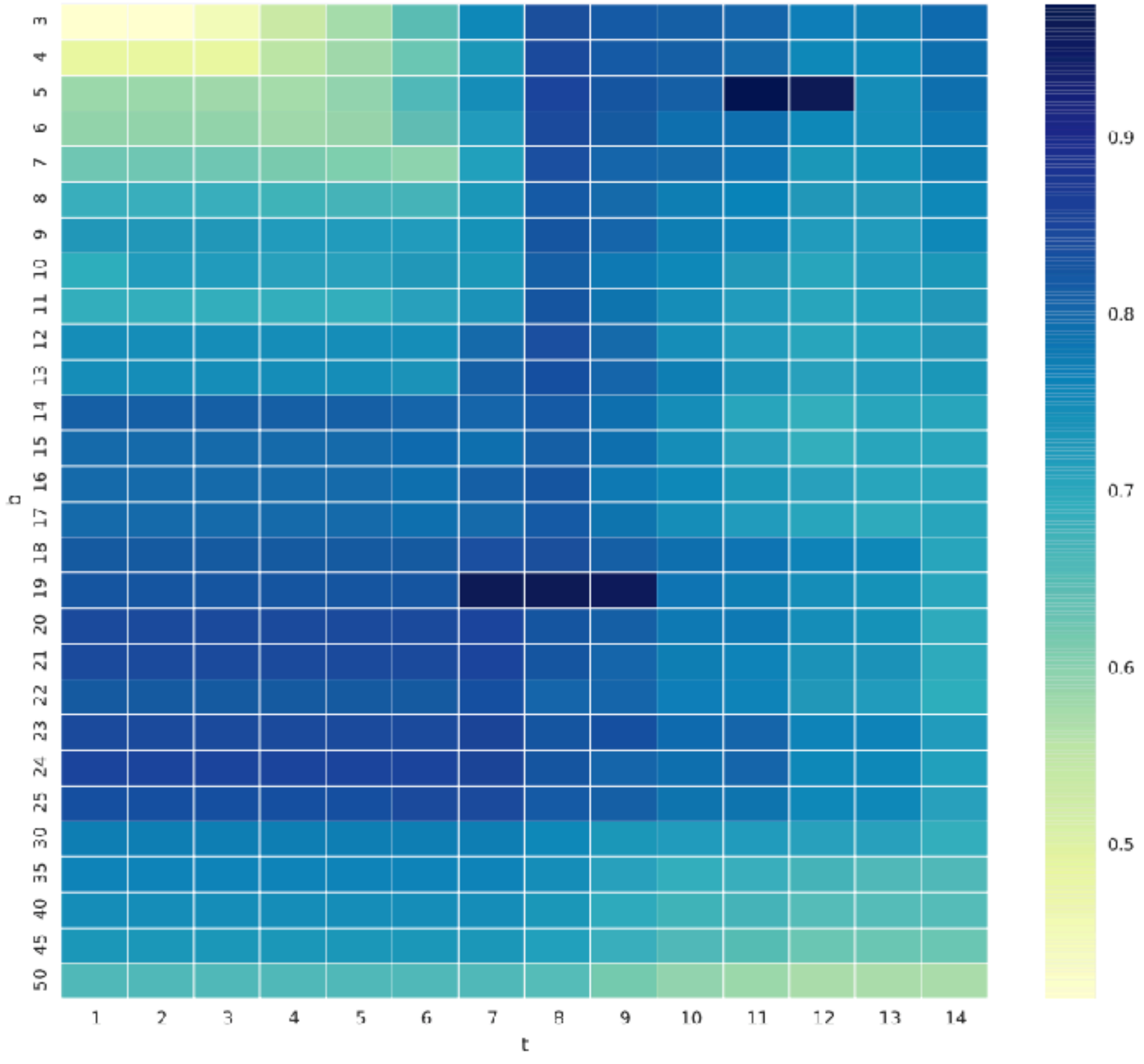
Optimal Configurations

CCFX's Precision vs. Recall

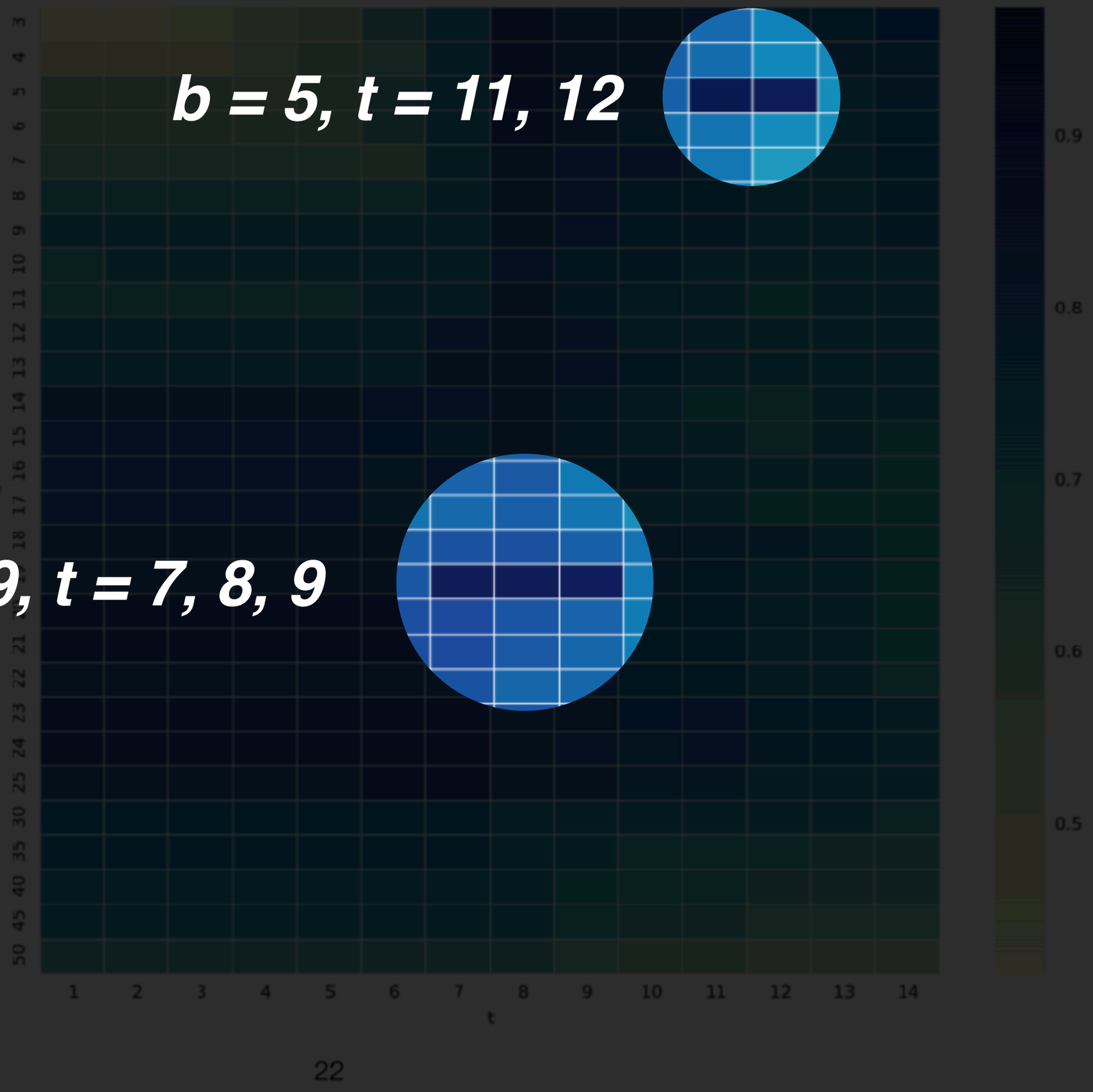


Measure	Value	ccfx's params	
		b	*t*
Precision	1.00	19	7, 8, 9
Recall	0.98	5	12

CCFX Optimal Config.



CCFX Optimal Config.



Tool	Settings	T	Settings	T
Clone det.				
ccfx (C)*	b=5,t=11	36	b=15,16,17 t=12	25
deckard (T)*	mintoken=30 stride=2 similarity=0.95	17	mintoken=50 stride=2 similarity=1.00	19
iclones (L)*	minblock=10 minclone=50	0	minblock=40 minclone=50	19
nicad (L)*	UPI=0.50 minline=8 rename=blind abstract=literal	38	UPI=0.30 minline=5 rename=consistent abstract=condition	22
simian (C)*	threshold=4 ignoreVariableNames	5	threshold=4 ignoreVariableNames	26
Plag. det.				
jplag-java	t=7	19	t=12	29
jplag-text	t=4	14	t=9	32
plaggie	M=8	19	M=14	33
sherlock	N=4, Z=2	6	N=5, Z=0	22
simjava	r=16	15	r=25	46
simtext	r=4	14	r=12	17
Compression				
7zncd-BZip2	mx=1,3,5	45	mx=1,3,5	64
7zncd-Deflate	mx=7	38	mx=7	64
7zncd-Deflate64	mx=7,9	38	mx=7	64
7zncd-LZMA	mx=7,9	41	mx=7,9	69
7zncd-LZMA2	mx=7,9	41	mx=7,9	69
7zncd-PPMd	mx=9	42	mx=9	68
bzip2ncd	C=1..9	38	C=1,2,3,..,8,9	54
gzipncd	C=7	31	C=9	54
icd	ma=LZMA2 mx=7,9	50	ma=LZMA mx=1,3	84
ncd-zlib	N/A	30	N/A	57
ncd-bzlib	N/A	37	N/A	52
xzncd	-e	39	2,3 6,7,8,9,e	64 65
Others				
bsdiff*	N/A	71	N/A	90
diff (C)*	N/A	8	N/A	29
difflib	whitespace=false autojunk=false	28	autojunk=true whitespace=true	42
fuzzywuzzy	token_set_ratio	85	ratio	65
jellyfish	jaro_distance	78	jaro_distance	82
ngram	N/A	49	N/A	59
cosine	N/A	48	N/A	68

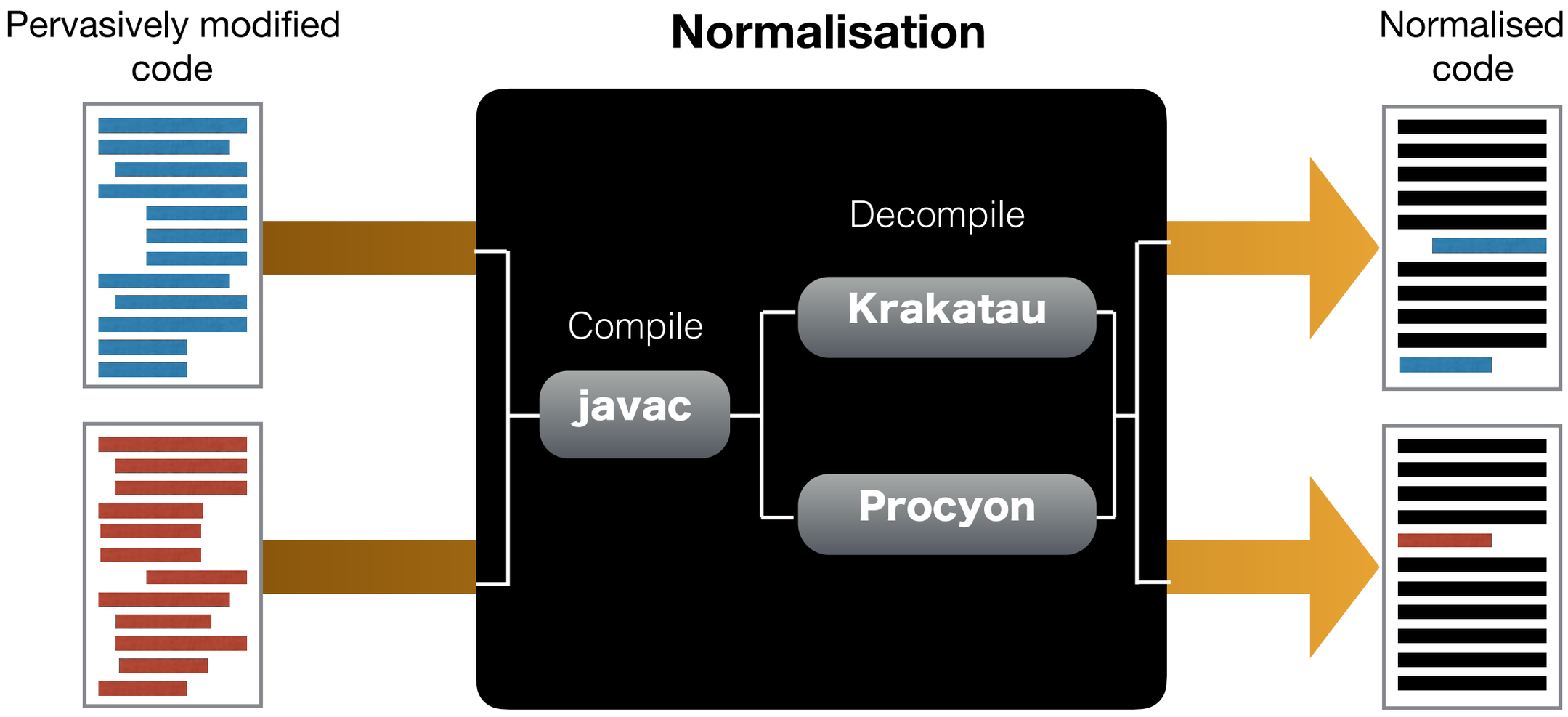
Pervasive Mod.

Boiler-Plate

The optimal configurations derived from one data set has a detrimental impact on the similarity detection results for another data set.

Cbuckley, Jpowell on en.wikipedia

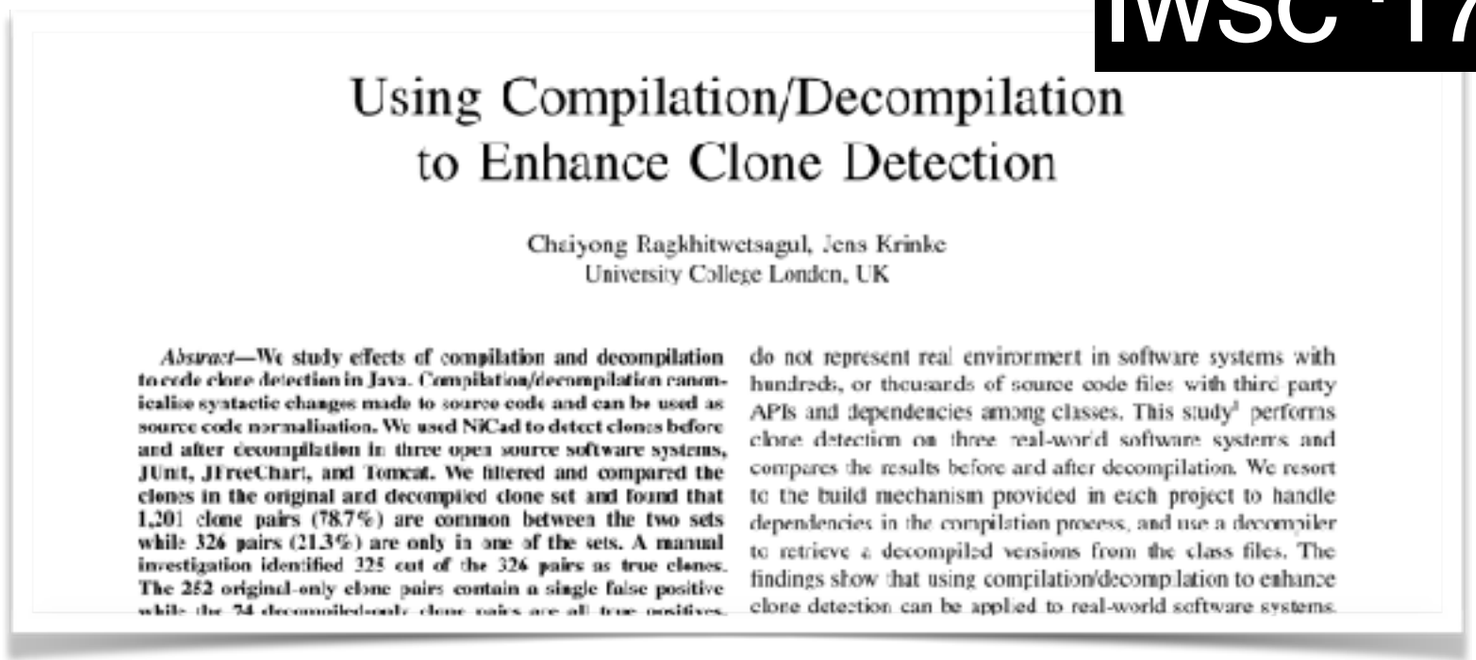
Normalisation by Decompilation




Clone det.
ccfx
deckard
iclones
nicad
simian
Plag det.
jplag-java
jplag-text
plaggie
sherlock
simjava
simtext
Comp.
7zncd-BZip2
7zncd-Deflate
7zncd-Deflate2
7zncd-LZMA
7zncd-LZMA2
7zncd-PPMd
bzip2ncd
gzipncd
icd
ncd-bzlib
ncd-zlib
xz-ncd
Others
bsdiff
diff
py-difflib
py-fuzzywuzzy
py-jellyfish
py-ngram
py-sklearn
Orig.
Dec.
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
F1

Compilation and decompilation can be used as an effective normalisation method that greatly improves similarity detection on Java source code (with statistical significance)

IWSC ‘17



Using Compilation/Decompilation to Enhance Clone Detection

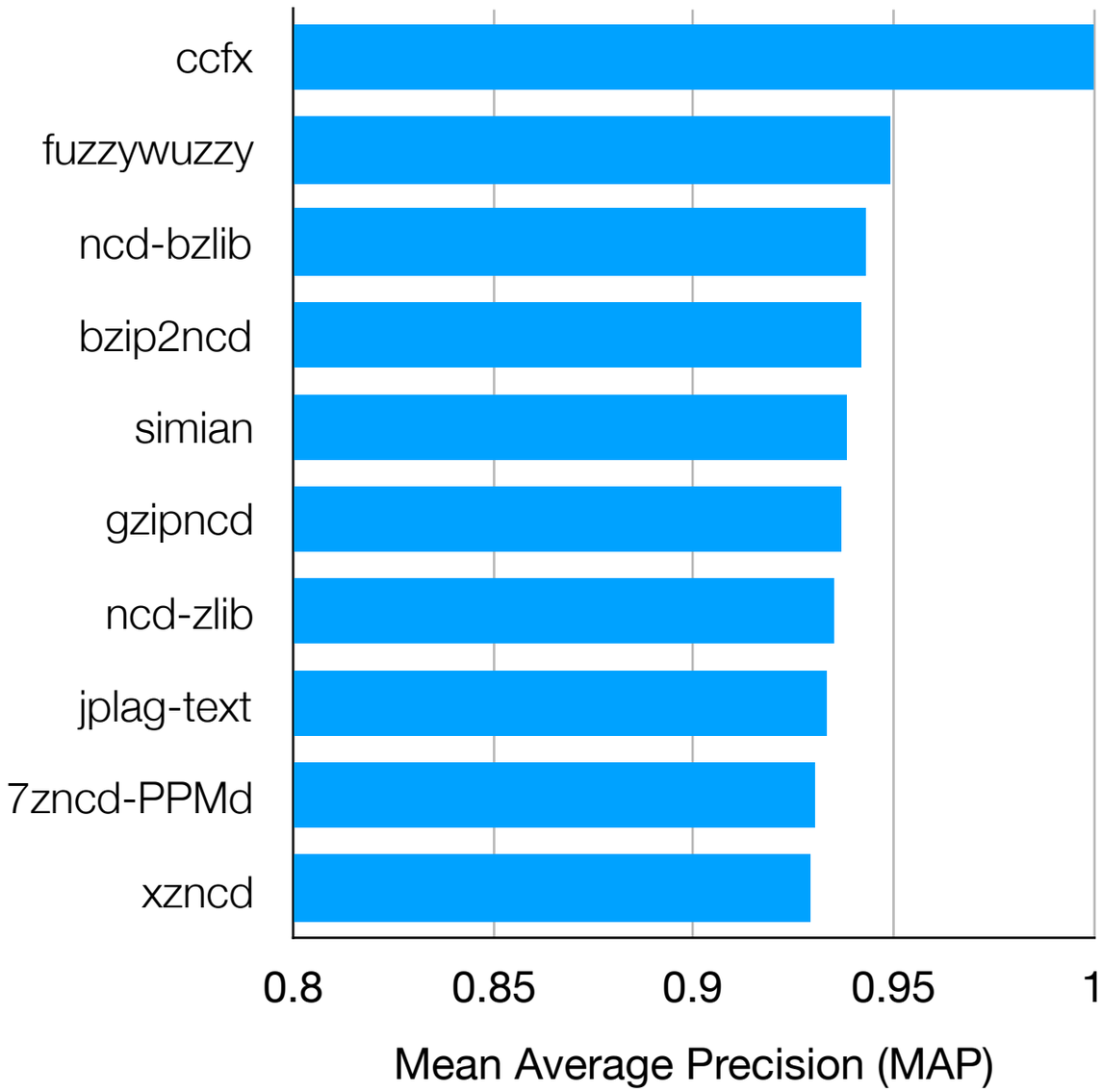
Chaiyong Ragkhitwetsagul, Jens Krinke
University College London, UK

Abstract—We study effects of compilation and decompilation to code clone detection in Java. Compilation/decompilation canonicalise syntactic changes made to source code and can be used as source code normalisation. We used NiCad to detect clones before and after decompilation in three open source software systems, JUnit, JFreeChart, and Tomcat. We filtered and compared the clones in the original and decompiled clone set and found that 1,201 clone pairs (78.7%) are common between the two sets while 326 pairs (21.3%) are only in one of the sets. A manual investigation identified 225 out of the 326 pairs as true clones. The 252 original-only clone pairs contain a single false positive while the 74 decompiled-only clone pairs are all true positives.

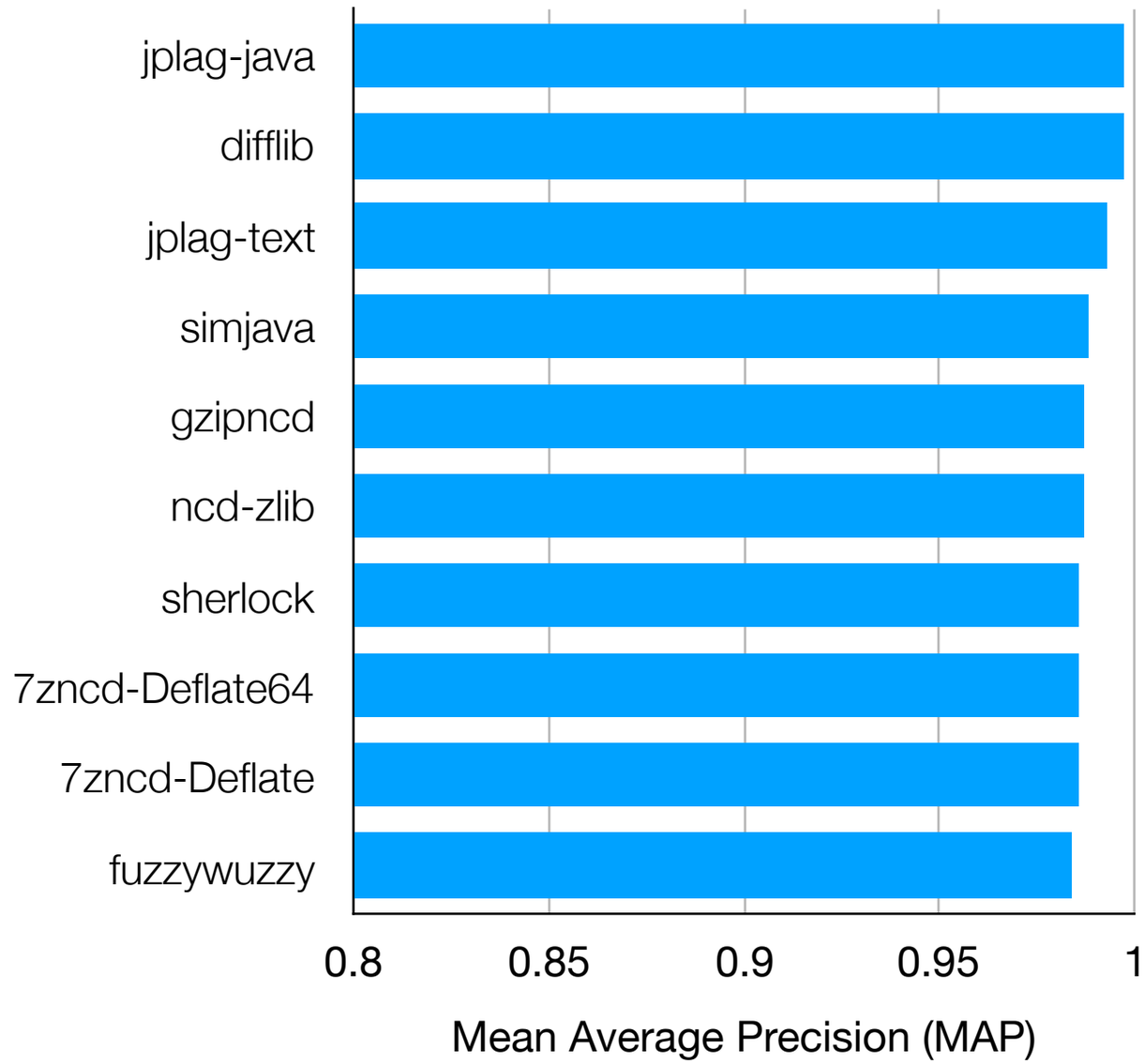
do not represent real environment in software systems with hundreds, or thousands of source code files with third party APIs and dependencies among classes. This study[1] performs clone detection on three real-world software systems and compares the results before and after decompilation. We resort to the build mechanism provided in each project to handle dependencies in the compilation process, and use a decompiler to retrieve a decompiled versions from the class files. The findings show that using compilation/decompilation to enhance clone detection can be applied to real-world software systems.

Ranked Results

Only Top *k* Results

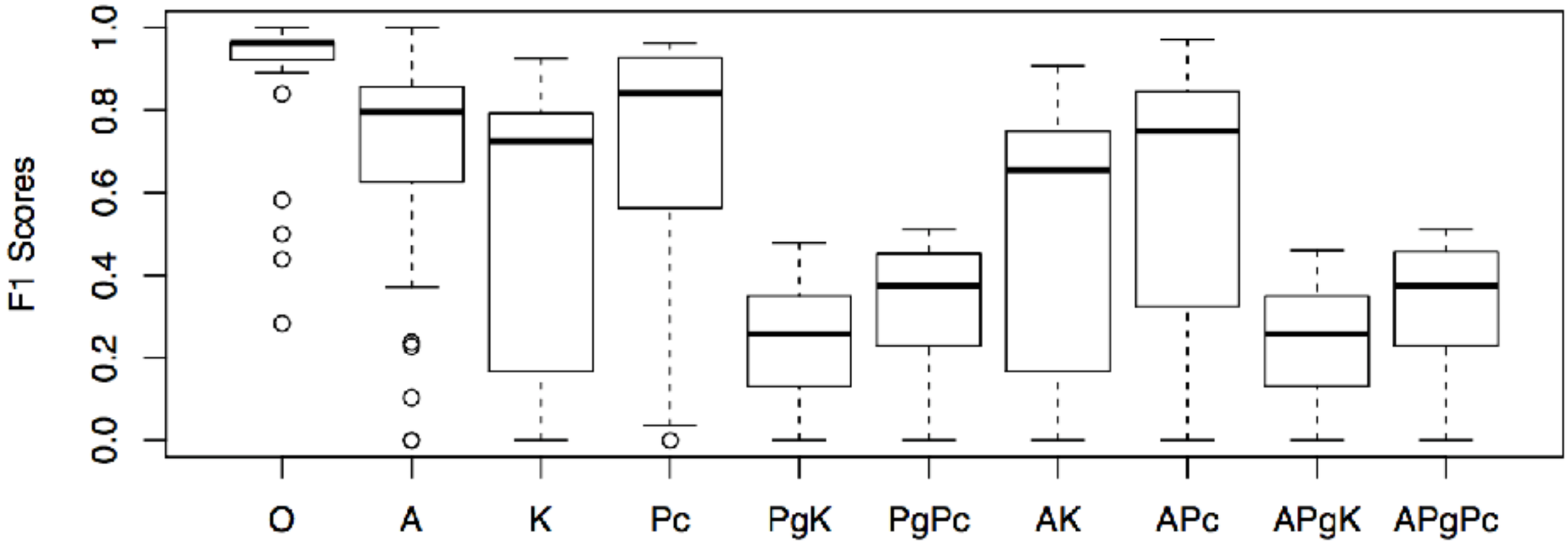


Pervasive Mod.



Boiler-Plate

Distribution of tool's F1 scores vs. pervasive mod. type



Original

O = original

Obfuscator

A = Artifice (source)

Pg = ProGuard (bytecode)

Decompiler

K = Krakatau

Pc = Procyon

F1 Score 0.1—0.4 0.4—0.6 0.6—0.8 0.8—1.0

Original

O = original

Obfuscator

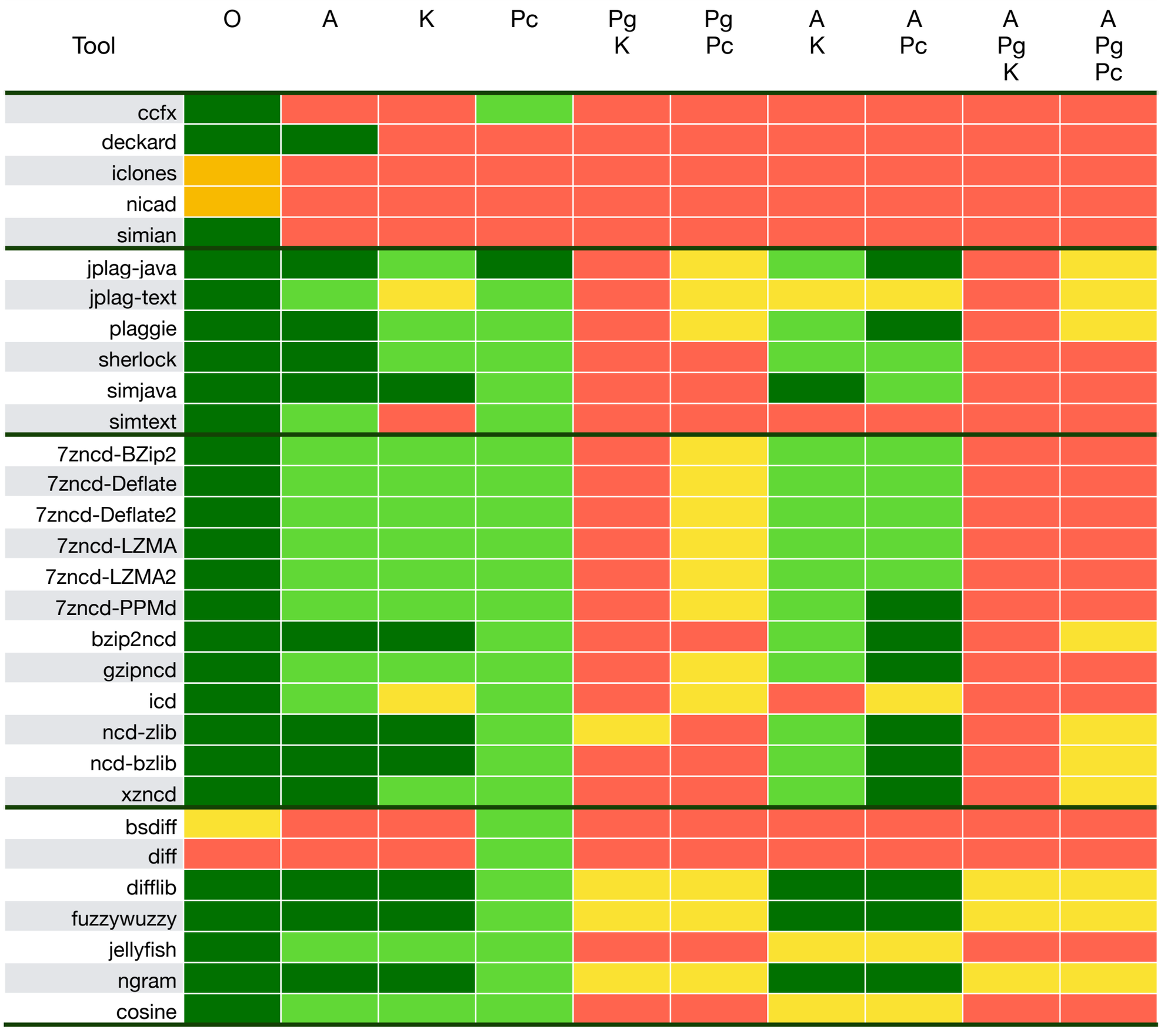
A = Artifice (source)

Pg = ProGuard (bytecode)

Decompiler

K = Krakatau

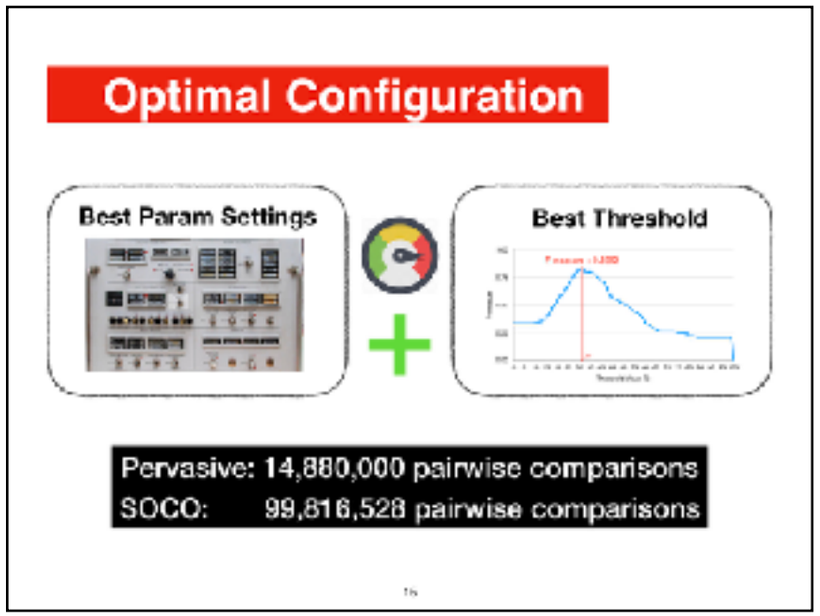
Pc = Procyon

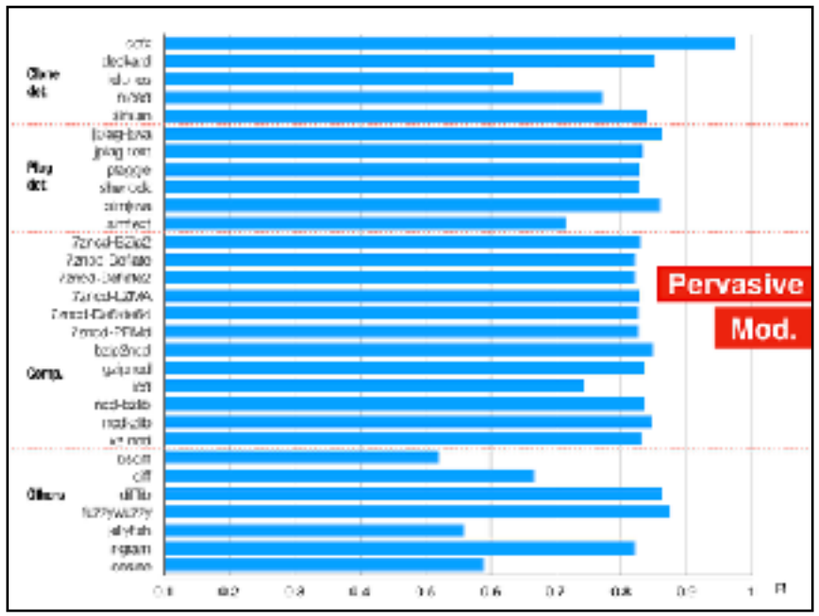


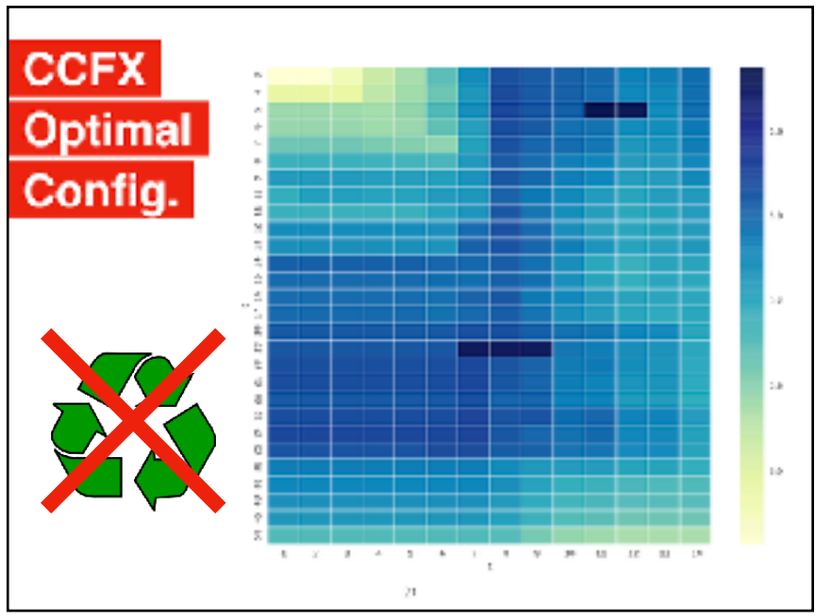
To Sum Up

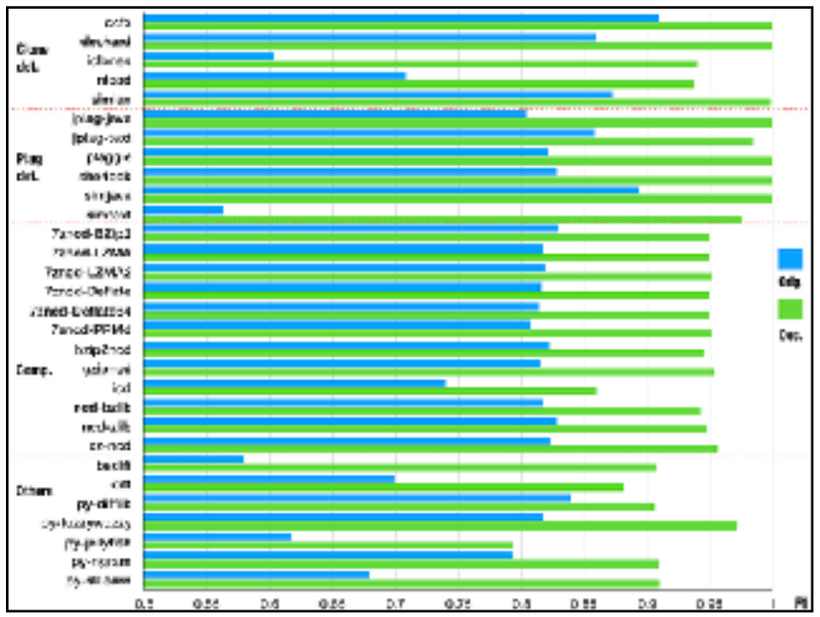
A Comparison of Code Similarity Analyzers

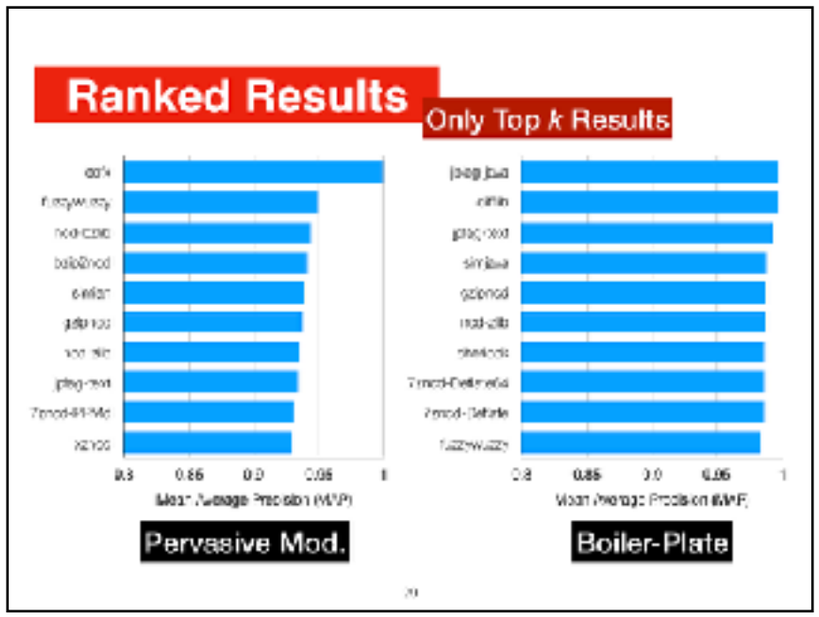


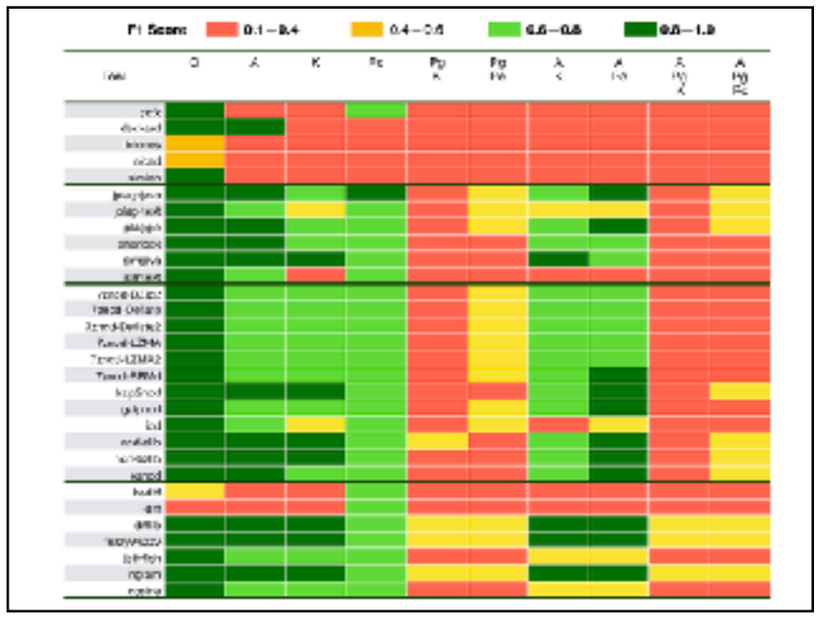












Research Note: http://www.cs.ucl.ac.uk/research/research_notes/

Website: http://crest.cs.ucl.ac.uk/resources/cloplag/