Clinical value of cerebrospinal fluid neurofilament light chain in semantic dementia

Meeter, Lieke H.H.; Steketee, Rebecca M.E.; Salkovic, Dina; Vos, Maartje E.; Grossman, Murray; McMillan, Corey T.; Niessen, Wiro J.; Papma, Janne M.; De Jong, Frank Jan; More Authors

DOI
10.1136/jnnp-2018-319784

Publication date
2019

Document Version
Final published version

Published in
Journal of Neurology, Neurosurgery and Psychiatry

Java Unit Testing Tool Competition - Sixth Round

Urko Rueda Molina
Research Center on Software Production Methods
Universitat Politècnica de València
Valencia, Spain
urueda@pros.upv.es

Fitsum Kifetew
Fondazione Bruno Kessler
Trento, Italy
kifetew@fbk.eu

Annibale Panichella
SnT – University of Luxembourg, Luxembourg
Delft University of Technology, The Netherlands
annibale.panichella@uni.lu

ABSTRACT

We report on the advances in this sixth edition of the JUnit tool competitions. This year the contest introduces new benchmarks to assess the performance of JUnit testing tools on different types of real-world software projects. Following on the statistical analyses from the past contest work, we have extended it with the combined tools performance aiming to beat the human made tests. Overall, the 6th competition evaluates four automated JUnit testing tools taking as baseline human written test cases for the selected benchmark projects. The paper details the modifications performed to the methodology and provides full results of the competition.

CCS CONCEPTS

• **Software and its engineering → Software defect analysis**; **Software testing and debugging**; **Empirical software validation**; *Search-based software engineering*;

KEYWORDS

tool competition, benchmark, mutation testing, automation, unit testing, Java, statistical analysis, combined performance

ACM Reference Format:
Urko Rueda Molina, Fitsum Kifetew, and Annibale Panichella. 2018. Java Unit Testing Tool Competition - Sixth Round. In *SBST'18: SBST'18:IEEE/ACM 11th International Workshop on Search-Based Software Testing , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3194718.3194728

1 INTRODUCTION

Continuing the tradition of the pastfi ve editions [8] of the Java unit testing tool competition, this year as well we have carried out the competition on a fresh set of test classes under test (CUT) selected from various projects. In the current edition, as in the previous [8], there are a total of four tools considered for the competition, namely: EvoSuite [1], JTexpert [12, 13], T3 [10, 11], and Randoop [6]. All the tools were executed against the same set of subjects, set of time budgets, and execution environment.

SBST'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5741-8/18/05…$15.00
https://doi.org/10.1145/3194718.3194728

In this year's edition, we do not have new tools entering into the competition, however the developers of EvoSuite and T3 have actively participated with improved versions of their tools. Furthermore, we have introduced a *combined analysis* in which we construct test suites by putting together all the tests generated, for a particular CUT, by all four tools. Such an analysis could shade some light on the overall strengths and weaknesses of the tools with respect to the CUTs under consideration. We also compare and contrast the results achieved by the various tools, either individually or combined together, against the manually-written test suites included in the original projects from which our test subjects were extracted.

For comparing the tools, we used well-stablished *structural coverage* metrics, namely *statement* and *branch* coverage, which we computed by using JaCoCo. Additionally, we apply *mutation analysis* to assess the fault revealing potentials of the test suites generated by the tools. To this aim, we use the PITest mutation analysis tool to compute the mutation scores of the various test suites (either automatically generated or manually-written).

Following lessons learned from previous editions, we have considered different time budgets, i.e., 10, 60, 120, and 240 seconds. Such a range of search budgets allows us to assess the capabilities of the tools in different usage scenarios. Furthermore, we augment the comparative analysis by considering the combined test suites composed of all the tests generated by all tools in the competition.

The report is organized as follows. Section 2 describes the set of benchmarks used this year, which were not used in previous competitions, and section 3 describes the objects under study (the tools) and the baseline (developer tests). Next, section 4 collects the changes introduced since the last competition [8]. The obtained results of running the competition are later described in section 5. Finally, our concluding remarks are available in section 6.

2 THE BENCHMARK SUBJECTS

Building a benchmark for assessing testing tools is always challenging as it requires considering different factors. For example, the benchmark should be a representative sample of real-world software projects [2]; the projects should be open-source and cover different application domains [2]; the classes should not be too trivial [7] (e.g., classes with only branchless methods) and should have different types of input parameters. With the aim of taking into account these factors, for this edition we focused on the top 500 GitHub repositories that satisfy the following criteria: (i) having more than 4K stars on 01/01/2018, (2) can be built using Maven, and (3) contain JUnit 4 test suites. From this large pool of possible

Table 1: Characteristics of the projects in our benchmark

Project	#Stars	#CUTs	10s	4m	# Sampled CUTs
Dubbo	16.3K	235	39m	15.7h	9
FastJason	12.6K	217	36m	14.5h	10
JSoup	5.6K	248	41.3m	16.5h	5
Okio	4.6K	44	7.3m	2.9h	10
Redisson	4.4K	1,392	3.9h	92.8h	10
Webmagic	6.1K	162	27m	10.8h	5
Zxing	17.4K	268	44.7m	17.9h	10

candidates, we randomly sampled (through a script thatfi ltered the projects based on the criteria) the following seven projects:

- *Dubbo*[1]: is a large *remote procedure call* (RPC) and microservice framework written in Java. For the competition, we focused on the maven sub-module *dubbo-common*.
- *FastJason*[2]: is a Java library providing utilities to convert JSON string to an equivalent Java object and vice versa.
- *JSoup*[3]: is a Java library containing classes and methods for extracting and manipulating data stored in HTML documents using Document Object Model (DOM) traversal methods and CSS and jQuery-like selectors.
- *Okio*[4]: is a small Java library providing utilities to access, store and process binary and character data using fast I/O and resizable buffers.
- *Redisson*[5]: implements a Java client for `redis` and provides distributed Java objects and services, such as `List`, `Queue`, `Cache`.
- *Webmagic*[6]: is a multi-thread web crawler framework supporting all typical clawer activities, such as url management and web page content extraction. For the competition, we focused on two maven sub-modules, namely *webmagic-core* and *webmagic-extension*.
- *Zxing*[7]: is an open-source library that supports the decoding and the generation of barcodes (e.g., QR Code).

Table 1 summaries the main characteristics of the selected projects. The total number of CUTs in each project ranges between 44 (Okio) and 1,392 (Redisson) classes. Computing test cases for the full projects would take between 7 minutes (10 seconds budget per CUT) and nearly 93 hours (4 minutes budget).

Comparing the tool participants on the entire projects was clearly unfeasible due to very large amount of time the competition would require: (i) running each tool on each CUT multiple times, (ii) with different time budgets, (iii) collecting the corresponding performance metrics (among which, mutation score is very resource and time demanding) from each independent run. For these reasons, we randomly sampled few CUTs from each project as reported in Table 1.

For sampling the CUTs, we used the same procedure used in the previous edition of the contest [8] and leveraging the McCabe's cyclomatic complexity. First, we computed the cyclomatic complexity for all methods and classes in each project using the extended `CKJM` library[8]. Then, wefi ltered the benchmark projects by removing classes that contain only methods with a McCabe's cyclomatic complexity lower than three. The McCabe's cyclomatic complexity of a method m corresponds to the number of branches in m plus one (or equivalently, the total number of independent paths in the controlfl ow graph of m). Therefore, ourfi ltered benchmark contains only classes with at least one method with at least two condition points (i.e., with a complexity ≥ 3). Thisfi lter reduces the chances of sampling very trivial classes with either no branches or that can be fully covered with few randomly generated tests [7].

From thefi ltered benchmark, we randomly sampled few classes from each project as follows:fi ve classes from *JSoup* and *Webmagic*, nine classes from *Dubbo*, plus 10 classes from each of the remaining four projects. This resulted[9] in 59 Java classes, whose number of branches ranges between 4 and 2197, while number of lines ranges between 26 and 3091, and number of mutants produced by PIT ranges between 16 and 1023.

3 THE BENCHMARK OBJECTS

In this edition of the competition, a total of four tools are considered. The tools are the same as in the previous edition, with the exception that EvoSuite has been updated to a new version. T3 has introduced some changes by way of bugfi xes and improved competition interface implementation for better integration with the evaluation framework. The other tools, i.e., Randoop and JTexpert, remain the same as in the previous edition.

3.1 Baseline human made JUnit tests

As baseline, we use test suites generated by Randoop, as well as the manually written test suites of the CUTs available from their respective projects. Even though we use human test suites as baseline with the aim of giving an idea of how the automatically generated test suites fair with respect to human written tests, it is difficult to draw direct parallels between the two. Human written test suites are typically evolved and improved overtime, and it is usually hard to determine exactly how much (human) effort has been spent in producing each test suite.

Additionally, we use the JTexpert tool as baseline, because it is not updated since the last competition and the authors are not actively participating in the current competition.

3.2 Competition Tools

This year, the active competitors are EvoSuite and T3. As shown in Table 2, EvoSuite uses an evolutionary algorithm for evolving test suites, while T3 employs a random testing strategy. The table also summarizes the main characteristics of the four tools considered in this edition. Moreover, similar to what was done in previous editions, participants were able to test their implementation using a set of sample CUTs. Concretely, the full set of CUTs from the previous competition [8]. Note that the CUTs used in this edition are all newly selected and were not revealed to the authors before running the competition.

[1] https://github.com/alibaba/dubbo
[2] https://github.com/alibaba/fastjson
[3] https://github.com/jhy/jsoup
[4] https://github.com/square/okio
[5] https://github.com/redisson/redisson
[6] https://github.com/code4craft/webmagic
[7] https://github.com/zxing/zxing

[8] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/
[9] https://github.com/PROSRESEARCHCENTER/junitcontest/tree/master/bin/benchmarks_6th

Table 2: Summary of tools

Tool	Technique	Static analysis
EvoSuite [1]	Evolutionary algorithm	yes
JTexpert [12, 13]	Guided random testing	yes
T3 [10, 11]	Random testing	no
Randoop [6]	Random testing	no

4 CONTEST METHODOLOGY

This 6th contest shares most of the methodology from the previous edition [8]. We will focus on describing the modifications made to run this year competition.

→ **Public contest repository**[10]**.** The full contest infrastructure was published to GitHub four months before the competition. The initial objective was to attract new participants raising the awareness, with no success. However, the long run aim was to share the competition experiences to allow future competitors to collaborate, better prepare their tools for automation, report bugs, request new features or improvements, etc. Therefore, the global goal was to advance the maturity of the infrastructure built for the competition, and so the efficiency and effectiveness of the tools by offering a public benchmark to compare with.

→ **JUnit tools set up**. Participants were able to test their correct operation for the contest using the latest version of the infrastructure. We provided them the full set of benchmarks from past 5th contest [8], which did not contain any of the benchmarks from this edition.

→ **CUTs**. We selected 59 new benchmark classes as described in Section 2 and that constitute the subjects of the competition.

→ **Execution frame**. This year a total of 5664 executions have been scheduled (5796 executions in the previous edition): 59 CUTs x 4 tools x 4 time budgets x 6 repetitions for statistical analyses. In an attempt to foster the replicability of the contest executions we have transferred the know-how of the past 5 years to a new environment operated by new people. We have switched the infrastructure from an HP Z820 workstation with two virtual machines, each with 8 CPU cores and 128GB RAM, to a cluster environment running Sun Grid Engine (SGE). We have used three physical nodes each with 24 CPU cores and 256GB RAM. On each node, we executed two repetitions of the tools on all four budgets, for a total of six repetitions in total. Similar to the previous edition, all tools were executed in parallel. For each tool, and each search budget, the contest infrastructurefi rst invokes the tool on each CUT to generate the test cases. Once test generation is completed, the infrastructure continues to the second phase, which is the computation of the metrics, i.e., coverage and mutation scores.

→ **Test generation**. The execution frame used this year granted enough power to repeat the generation of tests by tools a total of 6 times, to account for the inherent randomness of the generation processes. The tools had to compete for the available resources as they were run in parallel. The benchmark subjects used this year made the contest execution to sporadically hang during tests generation. In that case, we had to force kill some of the executions as neither the tools nor the contest infrastructure did succeed to stop the related processes. The impact for these executions is a 0 score as the provided budget is exceeded. Additionally, and continuing the automation procedure of past competitions, the CUTs were specified with the paths for: i) source javafi les, ii) compiled class files, and iii) the classpath with the required dependencies. However, this specification missed some critical dependencies on some CUTS (e.g. DUBBO-2, WEBMAGIC-4) and tools could have generated crashing test cases.

→ **Metrics computation**. We kept the strict mutation analysis time window of 5 minutes per CUT, and a timeout of 1 minute for each mutant. Moreover, mutants sampling[11] is applied on the set of mutants generated by PITest. The rationale behind it is to reduce the computation time and provide results in a feasible amount of time. Moreover, recent studies [4] showed that random sampling is particularly effective despite its very low computational complexity compared to other mutant reduction strategies. Note that we applied the same set of sampled mutants to evaluate the test suites generated by different tools on the same CUT.

→ **Combined analyses**. To explore whether the combined tools' tests would outperform the developer designed tests, we have introduced the combined analyses to evaluate the cooperative test performance. The process consists of building new test suites that contain all test cases generated by all tools on a given CUT and with a given time budget. Then, the metrics computation is performed on the combined test suite in the exact same way as for the individual tools. Yet, the computation costs increase to the sum of the costs required to evaluate the test suites generated by each individual tool. We approached it in a separate analysis to measure the achieved instruction and branch coverages, and the test effectiveness. Furthermore, due the high computation costs for the full combined analyses we were only able to obtain data on the budget of 10 seconds as we run out of time to compute the rest of the budgets.

→ **Time budgets**. In the former edition of the competition [8], we did not observe any significant improvement (i.e., coverage and mutation score) after four minutes of search budget. Therefore, for this edition of the competition we have decided to consider only four search budgets, i.e., 10, 60, 120 and 240 seconds. This allowed us to use the saved computation resources for the combined analyses introduced in this edition.

→ **Statistical Analysis**. Similar to the previous edition of the competition [8], we used some statistical tests to support the results collected in this edition of the competition. First, we use the Friedman test to compare the scores achieved by the different tools over the different CUTs and different time budgets. In total, each tool produced (59 CUTs × 4 budgets) = 236 data points, corresponding to the average scores achieved across six independent repetitions. Second, we applied the post-hoc Conover's test for pairwise multiple comparisons. While the former test allows us to assess whether the scores achieved by alternative tools differ statistically significantly from each other, the latter test is used to determine for which pair of tools the significance actually holds.

In this edition, we augmented the statistical analysis by using the *permutation test* [3] to assess whether there exists a significant interaction among the scores produced by the tools, the cyclomatic

[10] https://github.com/PROSRESEARCHCENTER/junitcontest

[11] We applied a random sampling of 33% for CUTs with more than 200 mutants, and a sampling of 50% for CUTs with more than 400 mutants

complexity of the CUTs and the allocated search budgets. The permutation test is a non-parametric equivalent of the ANOVA (Analysis of VAriance) test and it is performed by randomly permuting data points across different groups in a given distribution. For this test, we set the number of iterations to a very large number (i.e., 10^8) to obtain robust results [7].

Note that for all aforementioned statistical tests, we used the confidence level α=0.05; p-values obtained with the Conover's test were further adjusted with the Holm-Bonferroni procedure, which is required in case of multiple comparisons.

4.1 Threats to Validity

Conclusion validity. As in previous editions, we perform statistical analyses for significance. In addition, we applied the permutation test to analyze possible co-factors that could potentially influence the performance of the tools.

Internal validity. We have a trajectory of six competitions where the contest infrastructure has been constantly stressed and improved. Furthermore, this year the infrastructure was made public four months before the competition, which allowed for the identification of potential threats of implementation. Only the benchmarks to be used in the competition were hidden to the participants, while they were able to test the environment with the full benchmarks from past competition.

Construct validity. The scoring formula used to rank the tools —which is identical to the past edition— assigns a higher weight to the mutation coverage. Also, we apply a time window of 1 minute per mutant and a global timeout of 5 minutes per CUT, as well as a random sampling of the mutants to reduce the costs of metrics computation. Note that the set of sampled mutants for each CUT is kept the same for all tools and search budgets.

Additionally, killing hang processes during test generation did not directly impact on the metrics in the sense that they exceeded the budget and got 0 scores and 0 coverages for the related combinations of CUTs, budgets and runs. However, it might have indirectly affected the results as the competing factor for the available resources (all tools were run in parallel) would be influenced by the human factor (the time a hang execution is detected and terminated). Nonetheless, this threat is mitigated by the effect of the six repetitions for statistical significance.

External validity. To mitigate the effects on the low sampling of subjects (the benchmarks) and objects (the competition tools) we continuously introduce new software projects (which compose the benchmarks' CUTs) of varied complexity to account for their representativeness on the real world, and include developers' tests and a random test generator *Randoop* as baseline to compare the performance of the competing tools. Nonetheless, the representativeness of the testingfi eld is weak as most of the tools have a random nature and only *Evosuite* implements Search-Based Software Testing techniques. We expect so that the public contest repository could help to attract more tools from thefield.

5 CONTEST RESULTS

Following the detailed results [9] for the last contest, we provide the average results for each tool across six independent runs for budgets of 10 seconds (Table 8), 60 seconds (available online [5]), 120 seconds (available online [5]) and 4 minutes (Table 9). Table 3 summarizes the average (mean) instruction coverage (cov_i), branch coverage (cov_b), and strong mutation coverage (cov_m) achieved across all CUTs over different search budgets. This table also shows the overall scores, which are computed as the sum of the average scores reached across all CUTs and for each search budget, separately. As expected, the coverage metrics and the scores increases when larger time is given for test generation. This is true for all tools except: i) Randoop, for which the coverage metrics remains mostly unchanged and ii) T3 for budget 10s, which better manages the provided search budget (Table 8) while other tools exceed the budget at some extent, suffering from the scoring formula penalty (half coverage[12] score in the worst case according to [8]).

Table 3: Average (mean) coverage metrics and overall (sum) scores obtained across all CUTs.

Tool	Budget (in sec.)	cov_i			cov_b			cov_m			Score	Std.dev
		Min	Mean	Max	Min	Mean	Max	Min	Mean	Max		
evosuite	10	0.00	0.47	0.97	0.00	0.40	0.96	0.00	0.29	0.94	89.20	17.84
t3		0.00	0.41	1.00	0.00	0.35	0.97	0.00	0.27	0.99	124.51	16.40
jtexpert		0.00	0.36	0.99	0.00	0.30	0.98	0.00	0.34	1.00	93.19	16.82
randoop		0.00	0.37	0.98	0.00	0.28	0.94	0.00	0.26	0.97	99.20	3.33
evosuite	60	0.00	0.54	0.99	0.00	0.48	0.97	0.00	0.42	0.96	185.18	24.71
t3		0.00	0.47	1.00	0.00	0.42	1.00	0.00	0.31	0.99	146.43	16.13
jtexpert		0.00	0.39	1.00	0.00	0.34	0.98	0.00	0.36	1.00	136.79	15.36
randoop		0.00	0.38	1.00	0.00	0.31	0.97	0.00	0.27	1.00	118.49	3.04
evosuite	120	0.00	0.56	1.00	0.00	0.50	0.99	0.00	0.44	0.97	194.96	18.54
t3		0.00	0.49	1.00	0.00	0.44	1.00	0.00	0.33	0.99	153.96	16.05
jtexpert		0.00	0.39	1.00	0.00	0.34	0.98	0.00	0.35	1.00	135.16	17.70
randoop		0.00	0.39	1.00	0.00	0.31	0.96	0.00	0.27	1.00	119.05	3.47
evosuite	240	0.00	0.57	1.00	0.00	0.52	0.99	0.00	0.45	0.99	201.64	18.26
t3		0.00	0.49	1.00	0.00	0.44	1.00	0.00	0.32	0.99	154.55	13.88
jtexpert		0.00	0.41	1.00	0.00	0.36	0.98	0.00	0.37	1.00	144.14	15.00
randoop		0.00	0.39	1.00	0.00	0.32	0.98	0.00	0.26	1.00	120.47	1.75

Comparison with manual and combined suites. Table 7 compares the performance of manual tests written by the original developers and the performance of combined test suites —i.e., the test suites obtained as the union of the test generated by all tools on the same CUTs— using a 10s of search budget. For this analysis, we consider only the 49 CUTs for which we couldfi nd manually-written tests (i.e., excluding the project *redisson* that misses working manual tests for the selected CUTs). It is worth remarking that *DUBBO-2* is a special case since its configuration missed critical dependencies (e.g., `javassist`). and all tools failed to generate compilable tests.

To better understand our results, Figure 1[13] and Table 6 show the performance on 49 CUTs[14] of (1) each individual tool , (2) the performance of the combined suites with 10 seconds budget, and (3) manually-developed test cases from the projects' developers. The performance of the combined suites has been computed by building one test suite per CUT and run consisting of all the test cases generated by the four tools in each corresponding CUT and run (only tests generated with a budget of 10 seconds). The performance is analyzed by: (1) the achieved instruction coverage (cov_i), (2) the branch coverage (cov_b) and (3) the mutation coverage (cov_m).

The mainfi nding of this analysis is that combining the test cases generated by individual tools can outperform the human developed tests, for nearly all software projects selected as benchmarks in this competition. This is particularly interesting if we consider that the combined suites are built by combining the tests generated in only

[12] instruction, branch and mutation coverages

[13] H.axis = budgets, V.axis = coverage percents, 5th_manual = 5th contest dev. tests

[14] REDISSON had no working manual tests available from the project

Table 4: Overall scores and rankings obtained with the Friedman test. For this analysis, we consider all 59 CUTs.

Tool	Budget	Score	Std.dev	Ranking
EvoSuite	*	687	50.33	2.02
t3	*	580	22.52	2.38
jtexpert	*	513	10.10	2.57
randoop	*	457	13.90	3.03

Table 5: Results of the pairwise comparison according to the post-hoc Conover's test and considering all 59 CUTs

	EvoSuite	jtexpert	randoop	t3
EvoSuite	-	-	-	-
jtexpert	5.9×10^{-5}	-	-	-
randoop	5.2×10^{-5}	0.947	-	-
t3	0.097	0.051	0.051	-

10 seconds of search budget. Therefore, larger budgets would likely result in better performance for the combined test suites over the individual tool results and the human-developed tests.

Final scores and statistical results. Table 4 shows the overall scores achieved by the four tools at different search budgets together with the ranking produced by the Friedman test. According to this test, the four tools statistically differ in terms of scores across all 59 CUTs (p-value $< 10^{-12}$). To better understand which pairs of tools statistically differ, Table 5 reports the p-values produced by the post-hoc Conover's procedure. Note the p-values are adjusted with the Holm-Bonferroni correction procedure as required in case of multiple pairwise comparison. We note that `evosuite` achieves significantly higher scores than `jtexpert` and `randoop` while there is no (or only marginal significance) with `t3`. On the other hand, `t3` has marginally significantly higher scores than `jtexpert` and `randoop`. Finally, the remaining two tools (i.e., jtexpert and randoop) turn out to be statistically equivalent.

The permutation test reveals that there is a significant interaction between the achieved scores and the tools being used to generate the tests (p-value$< 10^{-16}$), further confirming the results of the Friedman test. There is a significant interaction between the performance scores, the McCabe's cyclomatic complexity of the target CUTs and the testing tools (p-value$< 10^{-16}$). In other words, the scores of the alternative testing tools significantly differ for very complex CUTs (i.e., with large cyclomatic complexity). Moreover, the interaction between the testing tools and the adopted search budgets statistically interact with the achieved performance score (p-value$< 10^{-16}$). This means that the scores of the tools significantly increase when using larger search budgets.

6 CONCLUDING REMARKS

The combined tools performance analysis introduced in this edition reveals the power of a "*super-tool*" built over individual testing tools, which can potentially outperform developer tests, even with a small search budget like 10 seconds. This scenario brings an interesting field to explore in future competitions. Instead of running the tools isolated from each other, they could cooperate by trying to generate better test cases (more effective) in less time (more efficient).

Table 6: Comparison with manually-written tests and combined suites for 49 CUTs (without REDISSON)

Tool	Budget(s)	cov_i	cov_b	cov_m
`combined`	10	70.94	62.91	53.54
`manual`	-	54.16	46.37	34.87
`evosuite`	240	66.90	60.77	53.00
`t3`	240	53.18	48.29	38.78
`jtexpert`	240	48.35	42.88	43.41
`randoop`	240	41.28	34.56	26.41
`* (4 tools)`	10	43.92	36.81	32.39

Table 7: Results for manual and averaged combined results

	manual (49 CUTs)			combined 10 seconds (49 CUTs)		
CUT	cov_i	cov_b	cov_m	cov_i	cov_b	cov_m
DUBBO-10	72.9	63.2	66.3	29.9	31.4	40.0
DUBBO-2	37.9	32.2	41.5	0	0	0
DUBBO-3	41.6	34.1	33.7	95.4	95.6	74.5
DUBBO-4	60.0	37.5	48.1	92.2	94.4	94.4
DUBBO-5	51.5	55.5	70.0	96.9	94.4	96.6
DUBBO-6	80.4	75.4	91.8	50.0	56.2	61.4
DUBBO-7	75.3	58.3	98.0	100.0	97.9	100.0
DUBBO-8	0	0	0	80.6	64.4	84.2
DUBBO-9	59.2	42.3	74.6	85.8	76.1	22.0
FASTJSON-10	24.4	25.0	17.2	60.0	50.0	58.6
FASTJSON-1	7.8	5.5	5.3	21.9	20.1	8.2
FASTJSON-2	55.3	46.6	62.3	56.0	48.9	52.8
FASTJSON-3	56.2	49.4	22.3	22.7	15.5	4.5
FASTJSON-4	85.9	86.6	71.8	80.8	56.1	31.2
FASTJSON-5	30.0	22.5	41.2	49.4	42.3	59.4
FASTJSON-6	1.4	0	0	66.4	55.6	59.6
FASTJSON-7	35.5	26.3	7.5	84.0	77.2	74.6
FASTJSON-8	57.3	45.8	37.7	82.5	72.2	32.6
FASTJSON-9	54.6	57.8	51.3	44.6	54.6	56.1
JSOUP-1	70.9	59.2	1.5	98.1	89.9	6.0
JSOUP-2	34.7	26.0	16.6	66.7	64.9	0
JSOUP-3	75.9	51.1	80.0	80.1	49.8	55.6
JSOUP-4	87.8	90.0	86.6	95.9	91.6	83.3
JSOUP-5	89.5	85.5	25.2	65.5	37.5	14.4
OKIO-10	83.6	66.6	42.1	79.5	86.6	80.7
OKIO-1	83.1	76.0	3.1	77.2	64.4	1.4
OKIO-2	89.0	85.2	51.7	89.6	95.4	98.2
OKIO-3	90.9	90.0	29.4	75.0	63.6	75.2
OKIO-4	89.5	72.8	27.3	22.8	19.2	17.6
OKIO-5	83.6	62.1	6.8	91.0	63.3	60.7
OKIO-6	62.8	32.1	70.0	78.5	76.1	81.6
OKIO-7	100.0	75.0	92.6	80.9	85.4	79.2
OKIO-8	97.0	88.6	74.5	73.7	62.8	47.3
OKIO-9	93.3	100.0	46.1	60.0	42.3	38.4
WEBMAGIC-1	0	0	0	32.0	19.0	41.6
WEBMAGIC-2	0	0	0	44.5	22.8	43.4
WEBMAGIC-3	24.7	0	4.2	100.0	79.6	100.0
WEBMAGIC-4	0	0	0	74.6	16.6	65.5
WEBMAGIC-5	0	0	0	95.1	95.0	93.7
ZXING-10	91.3	83.8	20.6	91.0	81.8	10.0
ZXING-1	0	0	0	44.6	34.7	45.2
ZXING-2	0	0	0	23.8	22.3	11.6
ZXING-3	88.6	66.1	40.5	98.8	91.3	99.0
ZXING-4	84.1	74.3	27.0	96.7	96.9	2.8
ZXING-5	93.5	90.0	25.8	99.8	98.7	17.2
ZXING-6	0	0	0	81.7	71.8	74.0
ZXING-7	72.4	60.5	1.0	100.0	100.0	100.0
ZXING-8	0	0	0	62.3	60.5	69.2
ZXING-9	80.9	73.4	95.7	97.8	96.2	100.0

ACKNOWLEDGMENTS

Special thanks to Wishnu Prasetya whose interest on the combined tools performance pushed us to make it a reality. This work has been partially supported by the Spanish Ministry of Economy and Competitiveness (MINECO) under the project DataME (TIN2016-80811-P); and by the H2020 EU project SUPERSEDE under agreement number 644018.

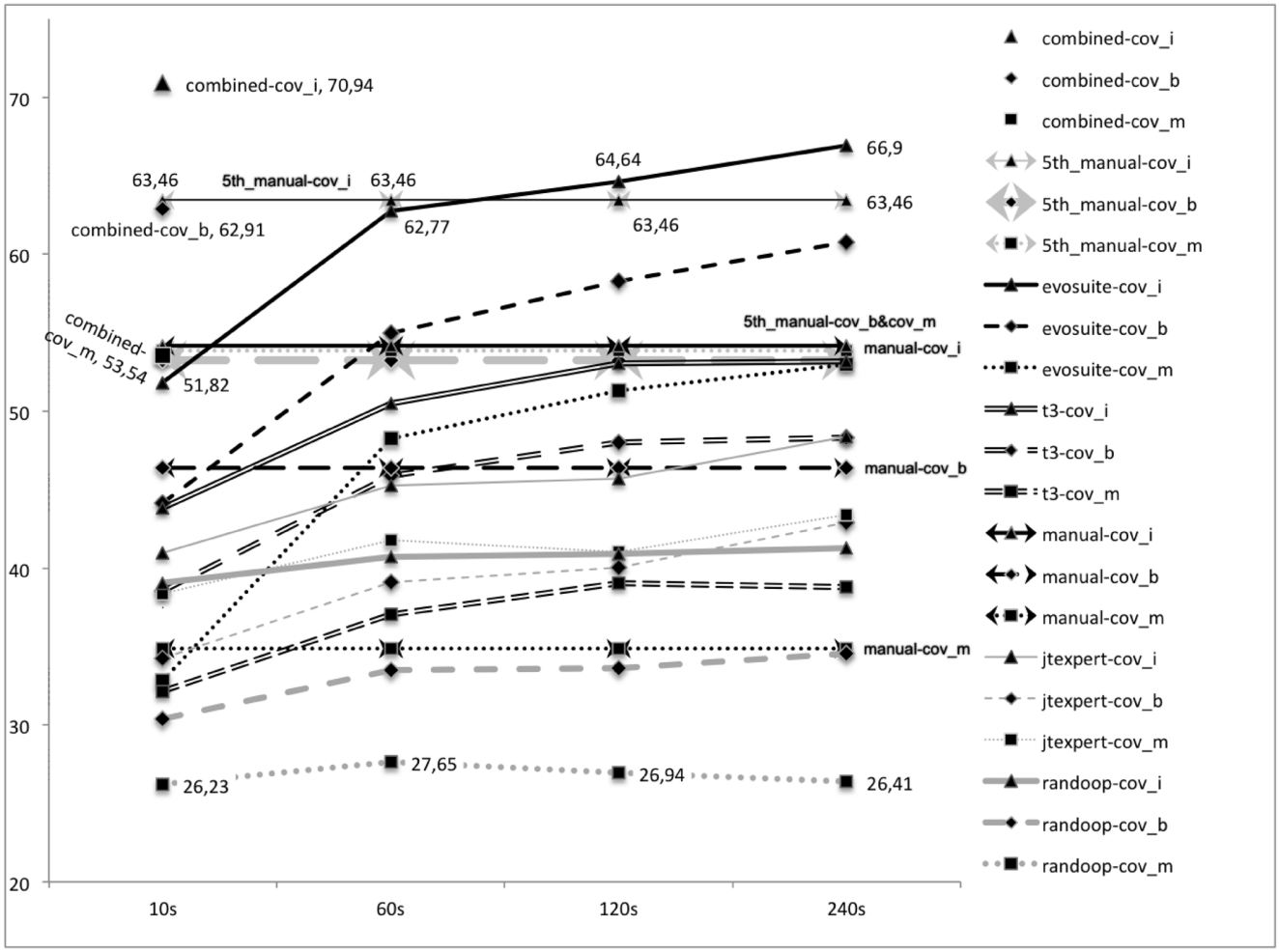

Figure 1: Test performance on 49 CUTs (without REDISSON)

REFERENCES

[1] A. Arcuri, J. Campos, and G. Fraser. 2016. Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, 401–408.

[2] Gordon Fraser and Andrea Arcuri. 2014. A Large Scale Evaluation of Automated Unit Test Generation Using EvoSuite. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 24, 2 (2014), 8.

[3] Phillip Good. 2013. *Permutation tests: a practical guide to resampling methods for testing hypotheses.* Springer Science & Business Media.

[4] R. Gopinath, I. Ahmed, M. A. Alipour, C. Jensen, and A. Groce. 2017. Mutation Reduction Strategies Considered Harmful. *IEEE Transactions on Reliability* 66, 3 (Sept 2017), 854–874. https://doi.org/10.1109/TR.2017.2705662

[5] Urko Rueda Molina, Fitsum Kifetew, and Annibale Panichella. 8th March, 2018. *Java Unit Testing Tool Competition - Sixth Round.* Technical Report. https://github.com/PROSRESEARCHCENTER/junitcontest/blob/master/publications/SBSTcontest2018_detailed_results.pdf

[6] Carlos Pacheco and Michael D. Ernst. 2007. Randoop: feedback-directed random testing for Java. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion (OOPSLA '07)*. ACM, New York, NY, USA, 815–816. https://doi.org/10.1145/1297846.1297902

[7] A. Panichella, F. M. Kifetew, and P. Tonella. 2018. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering* 44, 2 (Feb 2018), 122–158. https://doi.org/10.1109/TSE.2017.2663435

[8] A. Panichella and U. Rueda Molina. 2017. Java Unit Testing Tool Competition - Fifth Round. In *2017 IEEE/ACM 10th International Workshop on Search-Based Software Testing (SBST)*. 32–38. https://doi.org/10.1109/SBST.2017.7

[9] Annibale Panichella and Urko Rueda. 24th February, 2017. *Java Unit Testing Tool Competition — Fifth Round.* Technical Report. https://github.com/PROSRESEARCHCENTER/junitcontest/blob/master/publications/SBSTcontest2017_detailed_results.pdf

[10] I.S.W.B. Prasetya. 2015. T3i: A Tool for Generating and Querying Test Suites for Java. In *10th Joint Meeting of the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE)*. ACM. http://dspace.library.uu.nl/bitstream/handle/1874/321619/950.pdf?sequence=1

[11] Prasetya, I.S.W.B. 2016. Budget-aware random testing with T3: benchmarking at the SBST2016 testing tool contest. In *Proceedings of the 9th International Workshop on Search-Based Software Testing*. ACM, 29–32. http://dx.doi.org/10.1145/2897010.2897019

[12] Abdelilah Sakti, Gilles Pesant, and Yann-Gaël Guéhéneuc. 2016. JTExpert at the Fourth Unit Testing Tool Competition. In *Proceedings of the 9th International Workshop on Search-Based Software Testing (SBST '16)*. ACM, New York, NY, USA, 37–40. https://doi.org/10.1145/2897010.2897021

[13] A. Sakti, G. Pesant, and Y. G. Guéhéneuc. 2015. Instance Generator and Problem Representation to Improve Object Oriented Code Coverage. *IEEE Transactions on Software Engineering* 41, 3 (March 2015), 294–313. https://doi.org/10.1109/TSE.2014.2363479

Table 8: Averaged results for 6 runs on 10 seconds time budget

	Randoop						T3						Evosuite						jTexPert					
CUT	gen_t	cov_i	cov_b	cov_m	U	B	gen_t	cov_i	cov_b	cov_m	U	B	gen_t	cov_i	cov_b	cov_m	U	B	gen_t	cov_i	cov_b	cov_m	U	B
DUBBO-10	1.3	0	0	0	0	0	4.5	12.8	18.1	17.9	0	0	16.0	25.9	24.1	11.5	0	0	13.1	3.4	0	0	0	.9
DUBBO-2	1.3	0	0	0	0	0	2.8	0	0	0	0	0	4.1	0	0	0	0	0	16.0	0	0	0	1.0	0
DUBBO-3	12.2	65.1	60.9	66.4	0	0	4.5	95.3	94.3	91.1	0	0	15.7	88.6	87.0	93.9	0	0	2.9	0	0	0	0	0
DUBBO-4	11.8	68.3	50.0	33.3	0	0	4.4	68.3	66.6	55.5	0	.1	14.8	91.3	92.3	66.6	0	0	12.3	71.9	62.5	62.9	0	.2
DUBBO-5	11.6	96.9	94.4	86.6	0	0	4.5	96.9	94.4	87.7	0	0	14.9	96.9	94.4	68.3	0	0	12.4	95.4	77.7	96.6	0	.1
DUBBO-6	11.4	2.1	2.8	2.3	0	0	4.4	4.3	2.8	2.3	0	0	15.2	4.3	4.7	3.1	0	0	13.2	48.1	54.7	61.4	0	.1
DUBBO-7	11.7	97.9	86.8	97.4	0	0	4.4	100.0	96.8	98.7	0	0	15.0	94.1	80.2	77.8	0	0	11.3	77.4	66.3	80.1	0	.1
DUBBO-8	11.3	4.5	2.0	1.9	0	0	4.7	80.0	62.5	72.1	0	0	15.4	78.9	61.1	49.5	0	0	13.9	76.3	48.4	80.1	0	.1
DUBBO-9	12.2	76.9	64.3	66.0	0	0	4.6	79.3	70.1	72.4	0	0	16.9	80.2	67.8	38.0	0	0	15.2	78.0	67.6	68.9	0	.2
FASTJSON-10	12.4	6.6	0	3.4	0	0	4.4	55.9	45.8	37.9	0	0	15.5	42.2	40.0	21.8	0	0	12.8	60.0	50.0	58.6	0	.1
FASTJSON-1	1.4	0	0	0	0	0	12.1	3.2	2.7	1.0	0	0	0	0	0	0	0	0	9.2	12.4	11.3	5.2	0	.3
FASTJSON-2	11.7	38.0	28.8	31.8	0	0	4.9	38.3	30.8	26.1	0	0	15.9	7.0	5.6	7.1	.3	0	0	0	0	0	0	0
FASTJSON-3	11.7	17.4	10.8	1.5	0	0	14.6	0	0	0	0	0	6.9	3.3	1.7	.9	0	0	9.7	7.8	5.1	8.5	0	.4
FASTJSON-4	11.7	73.8	52.7	42.8	0	0	11.3	6.7	3.8	6.5	0	.2	18.3	40.0	20.0	34.2	0	0	16.4	49.7	28.8	33.1	0	.6
FASTJSON-5	11.3	1.1	.4	1.2	0	0	6.5	1.9	1.1	2.8	0	0	19.2	0	0	0	.5	0	14.4	49.4	42.2	59.4	0	.2
FASTJSON-6	11.9	11.1	13.5	9.0	0	0	4.4	10.3	9.4	6.5	0	0	15.6	15.0	18.4	8.7	0	0	13.8	68.6	57.2	64.8	0	.2
FASTJSON-7	11.8	76.8	69.7	74.6	0	0	6.1	64.5	52.6	50.2	0	.4	13.9	53.7	43.6	15.7	0	0	13.9	72.4	62.4	62.0	0	.3
FASTJSON-8	11.6	57.9	44.6	20.3	0	0	14.4	0	0	0	0	0	16.8	80.7	69.6	74.6	0	0	14.9	67.7	48.2	69.5	0	.3
FASTJSON-9	11.3	13.5	12.5	14.4	0	0	4.4	17.0	18.7	19.7	0	0	15.3	33.3	38.5	27.6	0	0	12.1	43.5	53.9	53.9	0	0
JSOUP-1	11.9	95.0	75.8	6.0	0	0	14.5	28.0	21.2	13.5	0	0	15.8	96.8	85.7	59.2	0	0	14.5	95.0	82.4	24.9	0	.2
JSOUP-2	11.3	21.0	23.2	0	0	0	4.6	59.0	59.0	5.4	0	0	15.6	54.7	49.7	36.3	0	0	12.0	25.2	26.1	17.3	0	.3
JSOUP-3	11.3	21.3	8.8	26.2	0	0	4.3	70.8	41.1	67.9	0	0	16.2	65.8	39.6	28.3	0	0	13.4	62.1	36.8	75.8	0	.2
JSOUP-4	11.5	95.1	90.0	80.0	0	0	11.3	24.7	20.0	16.1	0	0	16.6	72.3	48.3	31.6	0	0	6.2	16.6	16.6	16.6	.6	0
JSOUP-5	11.7	48.0	18.3	24.1	0	0	4.5	59.3	32.5	34.7	0	0	16.8	63.1	35.5	23.4	0	0	12.6	0	0	0	1.0	0
OKIO-10	11.6	67.3	60.0	47.3	0	0	4.2	73.4	80.0	57.8	0	0	15.2	79.5	83.3	76.7	0	0	12.0	0	0	0	1.0	0
OKIO-1	11.6	75.5	61.8	.6	0	0	14.5	11.5	7.7	5.1	0	0	3.3	10.0	7.4	1.6	0	0	9.0	0	0	0	1.0	0
OKIO-2	11.8	71.4	63.0	63.8	0	0	4.4	79.3	78.4	70.3	0	0	9.2	41.2	40.8	20.4	0	0	14.9	83.3	82.3	93.2	0	.1
OKIO-3	11.9	0	0	0	0	0	4.3	25.7	23.0	13.9	0	0	16.7	73.0	56.1	27.9	0	0	13.6	22.5	16.1	25.7	0	.5
OKIO-4	11.4	1.4	.6	.9	0	0	14.3	0	0	0	0	0	15.7	12.6	10.5	10.2	0	0	13.5	14.8	12.7	17.2	.1	.2
OKIO-5	11.4	2.8	1.5	1.3	0	0	11.5	56.5	32.8	33.1	0	0	18.6	89.4	61.3	45.6	0	0	14.2	0	0	0	.9	0
OKIO-6	11.5	34.2	32.1	36.6	0	0	14.5	25.7	22.6	20.5	0	0	16.6	77.6	73.2	80.5	0	0	2.8	0	0	0	0	0
OKIO-7	11.6	60.0	50.0	44.7	0	0	14.2	0	0	0	0	0	15.2	79.3	85.4	54.4	0	0	11.9	41.8	22.2	36.5	0	.3
OKIO-8	11.6	66.1	40.9	30.7	0	0	14.2	72.5	62.1	48.0	0	.2	6.2	22.5	13.6	6.8	0	0	12.3	0	0	0	1.0	0
OKIO-9	11.3	53.3	38.4	35.8	0	0	4.2	60.0	42.3	28.2	0	0	15.1	60.0	42.3	25.6	0	0	11.9	53.3	42.3	38.4	0	.3
REDISSON-10	11.6	88.9	91.6	88.5	0	0	8.4	77.6	59.0	0	0	0	9.0	29.3	31.8	8.5	0	0	10.5	19.3	15.5	21.5	0	.3
REDISSON-1	12.7	10.4	0	3.4	0	0	12.8	6.1	11.1	0	0	0	15.3	10.9	11.1	.9	0	0	12.3	0	0	0	0	2.0
REDISSON-2	12.6	7.6	0	0	0	0	5.2	7.6	0	0	0	0	17.3	29.4	39.5	0	0	0	11.8	0	0	0	0	2.0
REDISSON-3	12.1	0	0	0	0	0	10.1	0	0	0	0	0	0	0	0	0	0	0	10.3	0	0	0	0	.7
REDISSON-4	12.5	9.3	0	15.7	0	0	6.1	7.7	0	0	0	0	16.4	13.3	0	0	0	0	10.9	0	0	0	0	1.3
REDISSON-5	12.8	0	0	0	0	0	5.4	0	0	0	0	0	14.7	0	0	0	0	0	12.7	0	0	0	0	2.0
REDISSON-6	11.3	0	0	0	0	0	5.3	0	0	0	0	0	14.1	0	0	0	0	0	8.0	0	0	0	0	1.0
REDISSON-7	1.2	0	0	0	0	0	3.1	0	0	0	0	0	0	0	0	0	0	0	8.5	1.7	2.0	3.5	0	1.5
REDISSON-8	11.5	73.0	58.3	80.6	0	0	5.0	71.5	54.1	0	0	.2	17.0	73.0	58.3	80.6	0	0	13.1	67.3	45.8	73.6	0	.2
REDISSON-9	11.7	79.4	40.9	55.5	0	0	4.1	71.3	56.0	4.6	0	0	16.1	90.6	68.1	30.0	0	0	10.7	47.7	42.4	46.7	0	.1
WEBMAGIC-1	1.3	0	0	0	0	0	2.7	0	0	0	0	0	15.8	21.3	10.2	24.6	0	0	12.8	0	0	0	0	1.0
WEBMAGIC-2	1.3	0	0	0	0	0	2.6	0	0	0	0	0	6.0	17.2	8.6	17.3	0	0	3.0	0	0	0	0	0
WEBMAGIC-3	11.7	85.3	41.9	58.2	0	0	4.4	84.2	24.4	49.1	0	0	15.3	95.3	55.2	33.8	0	0	16.0	83.3	63.8	83.3	0	.3
WEBMAGIC-4	11.6	74.6	16.6	65.5	0	0	14.3	0	0	0	0	0	3.5	0	0	0	0	0	2.8	0	0	0	0	0
WEBMAGIC-5	11.6	80.6	75.0	75.0	0	0	5.2	76.8	78.3	76.0	0	0	3.3	11.8	8.3	11.4	0	0	12.5	86.0	68.3	81.2	0	.2
ZXING-10	11.9	38.0	31.6	25.0	0	0	4.9	60.9	59.1	47.4	0	0	16.5	86.5	68.3	54.0	0	0	14.2	81.4	70.7	84.7	0	.2
ZXING-1	11.9	8.7	3.3	2.2	0	0	4.4	20.8	14.6	5.0	0	0	17.8	40.8	29.7	4.3	0	0	13.0	8.0	3.7	4.8	0	.7
ZXING-2	11.4	.3	0	0	0	0	13.2	12.1	10.1	7.7	0	0	9.1	11.7	12.1	3.4	0	0	2.6	0	0	0	0	0
ZXING-3	11.5	62.5	47.4	23.4	0	0	6.0	97.3	84.8	67.7	0	0	15.3	97.3	87.5	3.9	0	0	13.0	90.3	73.4	94.4	0	.1
ZXING-4	1.3	0	0	0	0	0	4.4	92.5	93.0	19.8	0	0	15.5	89.0	86.1	82.0	0	0	13.5	43.1	21.9	31.6	0	.1
ZXING-5	13.0	10.9	7.5	1.1	0	0	4.4	65.4	82.0	42.4	0	0	14.4	48.0	38.3	44.5	0	.1	13.3	93.0	89.1	65.7	0	.2
ZXING-6	11.3	0	0	0	0	0	4.4	76.1	64.0	39.2	0	0	16.2	43.0	36.2	45.2	0	.1	13.3	.9	.4	1.0	0	1.0
ZXING-7	11.9	50.0	40.7	35.5	0	0	4.3	55.9	53.9	47.1	0	0	16.0	97.2	96.0	72.7	0	0	13.2	99.3	98.0	100.0	0	0
ZXING-8	11.5	5.1	3.3	1.9	0	0	4.4	36.5	46.1	29.1	0	0	15.2	58.1	54.4	31.0	0	0	11.8	27.7	23.3	23.0	0	.3
ZXING-9	12.1	77.7	63.2	46.8	0	0	6.1	88.3	84.3	82.2	0	0	15.0	88.8	89.7	52.4	0	0	12.2	97.8	91.1	100.0	0	.2

Table 9: Averaged results for 6 runs on 240 seconds time budget

	Randoop						T3						Evosuite						jTexPert					
CUT	gen_t	cov_i	cov_b	cov_m	U	B	gen_t	cov_i	cov_b	cov_m	U	B	gen_t	cov_i	cov_b	cov_m	U	B	gen_t	cov_i	cov_b	cov_m	U	B
DUBBO-10	1.3	0	0	0	0	0	36.6	15.6	22.4	23.3	0	0	149.5	28.9	28.2	20.8	0	0	197.2	3.4	0	0	0	.9
DUBBO-2	1.7	0	0	0	0	0	3.7	0	0	0	0	0	4.4	0	0	0	0	0	199.0	0	0	0	1.0	0
DUBBO-3	254.4	70.7	68.2	27.8	0	0	19.0	96.0	95.1	90.4	0	0	187.0	96.5	97.0	85.7	0	0	3.0	0	0	0	0	0
DUBBO-4	242.4	68.3	50.0	33.3	0	0	10.6	68.3	66.6	55.5	0	.1	151.4	93.0	95.1	92.5	0	0	196.9	77.4	79.8	71.6	0	.3
DUBBO-5	242.3	96.9	94.4	93.3	0	0	27.8	96.9	94.4	87.7	0	0	150.5	97.4	94.4	78.8	0	0	196.6	96.9	73.1	100.0	0	.1
DUBBO-6	242.3	2.1	2.8	2.3	0	0	8.1	4.3	2.8	2.3	0	0	147.2	4.3	4.7	2.3	0	0	196.6	51.4	59.7	63.9	0	.1
DUBBO-7	244.4	100.0	95.8	100.0	0	0	18.7	100.0	97.9	99.0	0	0	152.7	100.0	97.9	99.0	0	0	197.2	92.0	78.8	97.4	0	.1
DUBBO-8	241.4	4.5	2.0	1.9	0	0	234.9	89.6	81.9	75.3	0	0	161.6	84.6	77.6	78.3	0	0	198.1	80.3	60.7	83.9	0	.2
DUBBO-9	242.0	78.6	67.6	69.3	0	0	82.7	80.7	72.0	72.8	0	0	175.3	85.3	77.0	70.3	0	0	198.8	82.5	74.7	72.7	0	.1
FASTJSON-10	246.0	6.6	0	3.4	0	0	13.4	60.0	50.0	44.8	0	0	159.1	60.7	50.0	45.4	0	0	197.1	60.0	50.0	58.6	0	.1
FASTJSON-1	1.5	0	0	0	0	0	182.5	9.6	8.3	3.1	0	0	234.8	23.7	20.3	10.2	0	0	279.3	28.4	28.7	4.7	0	.2
FASTJSON-2	245.3	40.5	31.4	29.5	0	0	237.7	46.1	39.7	39.2	0	0	227.4	48.0	40.4	43.5	0	0	227.1	57.0	50.4	65.8	0	.2
FASTJSON-3	244.3	19.3	13.5	1.5	0	0	244.7	28.1	22.2	14.0	0	0	222.0	25.5	18.5	16.9	0	0	223.6	43.6	36.6	45.3	0	.4
FASTJSON-4	243.7	93.7	72.2	55.9	0	0	244.6	15.4	10.5	4.6	0	.2	227.6	55.1	30.5	53.9	0	.1	168.5	76.2	75.0	48.0	0	.5
FASTJSON-5	241.4	1.1	.4	1.2	0	0	244.5	3.4	1.9	4.7	0	0	165.4	28.8	21.0	17.5	0	0	198.1	49.8	42.5	60.0	0	.2
FASTJSON-6	246.9	11.1	13.5	9.0	0	0	9.8	11.4	10.8	7.4	0	0	157.6	35.5	34.4	19.0	0	0	198.0	76.2	64.4	73.6	0	.2
FASTJSON-7	254.6	79.1	70.6	48.9	0	0	237.1	81.4	71.3	75.5	0	.4	166.9	81.0	71.6	49.7	0	0	200.3	83.0	75.9	78.6	0	.3
FASTJSON-8	250.5	78.5	70.6	11.1	0	0	243.2	80.8	66.8	40.9	0	.1	198.1	87.6	77.2	73.0	0	0	200.4	81.2	71.0	90.9	0	.1
FASTJSON-9	241.5	13.5	12.5	14.4	0	0	18.0	19.7	18.7	22.3	0	0	148.3	33.3	39.0	39.4	0	0	196.4	43.7	53.6	53.9	0	0
JSOUP-1	242.3	95.0	76.1	6.0	0	0	244.5	77.8	55.0	36.9	0	0	180.7	98.2	94.8	87.4	0	0	223.4	97.1	89.2	20.3	0	.2
JSOUP-2	241.4	21.0	23.2	0	0	0	216.2	69.7	70.7	2.0	0	0	152.2	60.3	55.7	40.8	0	0	196.2	27.1	27.8	17.3	0	.2
JSOUP-3	241.4	21.3	8.8	26.2	0	0	34.7	76.6	46.4	71.2	0	0	202.8	75.7	52.0	49.7	0	0	198.0	74.5	44.4	84.5	0	.1
JSOUP-4	242.2	95.1	90.0	80.0	0	0	207.1	95.5	91.6	81.1	0	0	147.6	82.9	81.6	72.7	0	0	8.6	16.6	16.6	16.6	.6	0
JSOUP-5	245.4	49.0	18.8	27.4	0	0	234.8	69.3	44.6	20.5	0	0	169.6	81.3	56.4	50.0	0	0	197.1	0	0	0	1.0	0
OKIO-10	242.0	67.3	60.0	47.3	0	0	31.1	73.4	80.0	57.8	0	0	149.9	78.2	83.3	80.7	0	0	196.1	0	0	0	1.0	0
OKIO-1	242.5	82.6	77.8	.9	0	0	244.5	0	0	0	0	0	234.3	65.7	56.1	23.9	0	0	227.7	0	0	0	.9	0
OKIO-2	242.5	81.2	85.0	92.1	0	0	119.4	89.0	95.4	78.9	0	0	205.3	91.8	94.9	78.6	0	0	200.5	86.4	90.1	95.6	0	.1
OKIO-3	244.0	0	0	0	0	0	105.8	27.2	25.0	17.1	0	0	166.1	97.5	87.7	69.1	0	0	198.5	25.6	21.9	28.7	0	.4
OKIO-4	241.5	1.4	.6	.9	0	0	244.4	0	0	0	0	0	238.4	65.0	49.8	51.1	0	0	198.5	0	0	0	1.0	0
OKIO-5	241.3	2.8	1.5	1.3	0	0	245.2	37.9	22.4	20.3	0	0	211.4	98.0	93.4	88.3	0	0	221.3	0	0	0	1.0	0
OKIO-6	242.2	34.2	32.1	36.6	0	0	245.1	53.8	42.2	46.6	0	0	156.5	82.8	85.7	86.6	0	0	2.9	0	0	0	0	0
OKIO-7	243.5	81.8	79.1	67.0	0	0	225.4	84.2	88.8	69.9	0	0	149.5	97.2	97.2	79.2	0	0	196.2	41.8	22.9	36.5	0	.3
OKIO-8	242.4	69.1	65.9	23.5	0	0	244.3	79.4	65.1	64.3	0	.2	237.1	79.9	59.4	60.4	0	0	214.8	0	0	0	1.0	0
OKIO-9	241.3	53.3	38.4	35.8	0	0	4.4	60.0	42.3	28.2	0	0	146.5	60.0	42.3	35.8	0	0	196.2	53.3	42.3	38.4	0	.3
REDISSON-10	242.6	90.0	97.8	90.4	0	0	243.6	84.3	77.8	0	0	0	0	0	0	0	0	0	9.5	0	0	0	0	0
REDISSON-1	262.7	10.4	0	3.4	0	0	163.7	6.0	11.1	0	0	0	0	0	0	0	0	0	196.5	0	0	0	0	2.0
REDISSON-2	252.8	7.6	0	0	0	0	13.3	7.6	0	0	0	0	0	0	0	0	0	0	196.0	0	0	0	0	2.0
REDISSON-3	245.0	0	0	0	0	0	247.9	0	0	0	0	0	37.8	0	0	0	0	0	32.5	0	0	0	0	0
REDISSON-4	254.5	9.3	0	15.7	0	0	163.7	6.2	0	0	0	0	0	0	0	0	0	0	21.4	0	0	0	0	0
REDISSON-5	252.9	0	0	0	0	0	152.5	0	0	0	0	0	0	0	0	0	0	0	40.5	0	0	0	0	0
REDISSON-6	241.8	0	0	0	0	0	163.6	0	0	0	0	0	0	0	0	0	0	0	16.9	0	0	0	0	0
REDISSON-7	1.3	0	0	0	0	0	3.3	0	0	0	0	0	0	0	0	0	0	0	29.0	0	0	0	0	0
REDISSON-8	242.2	73.0	58.3	80.6	0	0	35.9	71.7	54.1	0	0	.2	0	0	0	0	0	0	18.8	0	0	0	0	0
REDISSON-9	242.5	80.8	45.4	58.3	0	0	42.4	89.7	68.1	4.6	0	0	170.9	99.5	99.2	74.5	0	0	21.9	50.0	50.0	31.9	0	.1
WEBMAGIC-1	1.3	0	0	0	0	0	2.7	0	0	0	0	0	154.2	33.1	22.4	42.0	0	0	197.1	0	0	0	0	.9
WEBMAGIC-2	1.3	0	0	0	0	0	2.7	0	0	0	0	0	239.6	59.1	41.5	53.6	0	0	2.9	0	0	0	0	0
WEBMAGIC-3	242.0	87.6	59.3	66.3	0	0	56.3	84.2	28.1	51.7	0	0	200.8	99.8	84.1	83.1	0	0	204.3	100.0	76.5	100.0	0	.4
WEBMAGIC-4	243.0	74.6	16.6	65.5	0	.1	244.3	0	0	0	0	0	3.6	0	0	0	0	0	3.0	0	0	0	0	0
WEBMAGIC-5	242.7	83.8	90.0	82.2	0	0	8.5	87.0	100.0	87.5	0	0	67.0	0	0	0	0	0	197.1	95.1	96.6	92.7	0	.2
ZXING-10	248.5	38.0	31.6	16.0	0	0	238.0	72.1	70.0	54.9	0	0	172.9	91.4	81.7	73.1	0	0	212.5	85.1	75.3	25.1	0	.1
ZXING-1	254.2	8.7	3.3	2.9	0	0	250.1	31.2	22.5	11.5	0	0	216.8	47.7	39.3	16.6	0	0	203.1	42.2	34.5	42.4	0	.8
ZXING-2	241.4	.3	0	0	0	0	244.6	4.4	4.4	2.2	0	0	232.6	31.5	30.3	32.9	0	.1	2.6	0	0	0	0	0
ZXING-3	241.8	62.5	47.4	24.3	0	0	13.9	97.7	85.8	67.8	0	0	163.7	97.9	91.6	68.6	0	0	207.2	97.5	82.4	98.9	0	.1
ZXING-4	1.3	0	0	0	0	0	23.0	92.8	94.9	18.6	0	0	171.3	95.9	95.3	84.3	0	0	209.8	65.4	47.1	37.0	0	0
ZXING-5	246.2	10.9	7.5	1.1	0	0	97.8	68.4	89.3	37.6	0	0	207.4	91.5	88.9	52.5	0	0	206.7	96.7	95.0	50.1	0	.1
ZXING-6	241.3	0	0	0	0	0	236.1	77.5	66.7	44.5	0	0	178.0	70.0	61.4	37.7	0	0	201.3	48.8	42.9	51.2	0	.7
ZXING-7	251.6	54.1	48.6	39.3	0	0	196.1	55.9	53.9	47.1	0	0	199.7	98.1	97.3	92.4	0	0	197.3	99.0	98.0	100.0	0	0
ZXING-8	242.7	5.1	3.3	1.9	0	0	13.8	39.7	54.4	33.9	0	0	151.4	80.1	82.2	46.7	0	0	196.3	34.4	29.4	23.0	0	.3
ZXING-9	244.6	77.7	63.2	46.8	0	0	17.9	94.1	93.8	85.4	0	0	147.9	98.4	96.9	63.4	0	0	197.2	100.0	93.5	100.0	0	.2