Do We Teach Useful Statistics For Performance Evaluation?

Lubomír Bulej, Vojtěch Horký, Petr Tůma

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics, Charles University
Malostranské náměstí 25, Prague 1, 118 00, Czech Republic

first.last@d3s.mff.cuni.cz

ABSTRACT

Basic topics from probability and statistics – such as probability distributions, parameter estimation, confidence intervals and statistical hypothesis testing – are often included in computing curricula and used as tools for experimental performance evaluation. Unfortunately, data collected through experiments may not meet the requirements of many statistical analysis methods, such as independent sampling or normal distribution. As a result, the analysis methods may be more tricky to apply and the analysis results may be more tricky to interpret than one might expect. Here, we look at some of the issues on methods and experiments that would be considered basic in performance evaluation education.

1. INTRODUCTION

In experimental performance evaluation, we often need to draw conclusions from measurements. Because measurements may suffer from artifacts such as noise, observation overhead or workload fluctuations, it is recommended that conclusions be derived using statistical methods – indeed, papers such as [1] demonstrate how careless evaluation distorts research results and call for statistically (more) rigorous experimental evaluation.

In our experience, one problem with this call is in choosing appropriate statistical methods. We may encounter measurements that exhibit long range dependence between samples, have unknown probability distributions with long tails, or are not stationary – generally, measurements with properties that complicate statistical analysis. Even in mundane situations, it is not easily possible to recommend a particular statistical method; instead, considerable expertise in applying statistical methods may be required.

The problem becomes acute when viewed through the optics of a typical bachelor level course in probability and statistics. Such a course would include basic statistical methods such as computation of confidence intervals or statistical hypothesis testing for some well behaved data sets – something our measurements are not. Obviously, a naive application of basic statistical methods to realistic measurements may not always provide rigorous conclusions.

In this paper, we want to take a closer look at the problem by investigating how basic statistical methods behave when applied to several realistic measurements. We are not after a comprehensive evaluation (given the variability in measurement scenarios, it is not clear whether a comprehensive evaluation is even possible), our goal is merely to provide the reader with some feel for how things can differ between the idealized settings that the chosen statistical methods assume and the realistic settings of performance evaluation experiments.

The paper is quite short. In Section 2, we briefly list the basic statistical methods we examine. In Section 3, we present our results. Concluding remarks close the paper in Section 4.

2. METHODS EXAMINED

Our idea is to examine the working of those statistical analysis methods that are taught among basic topics from probability and statistics in software related bachelor level courses. We focus on the use of confidence intervals for expressing performance, and the use of statistical hypothesis testing for comparing performance. In particular, we evaluate:

- computation of confidence intervals for sample mean with asymptotic normality assumptions,
- use of Welch t-test [13] and Mann-Whitney U-test [8] for detecting difference in performance.

To justify our choice, we look at which statistical analysis methods are mentioned in several sources of bachelor program information, starting with the ACM/IEEE curricula recommendations.

The guidelines for undergraduate computer science programs [5] are perhaps the least specific, suggesting that probability and statistics do not deserve a full core course: "while we do note a growing trend in the use of probability and statistics in computing ... we still believe it is not necessary for all CS programs to require a full course in probability theory for all majors."

More emphasis on probability and statistics can be found in the guidelines for software engineering programs [6], which explicitly support statistics among core topics: "the interac-

ICPE '17 Companion, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3053600.3053638

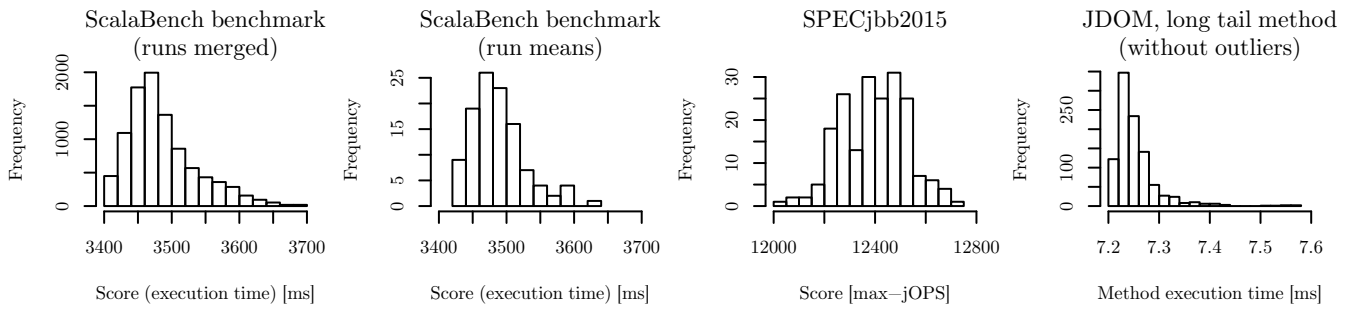


Figure 1: The measurement histograms collected in our experiments. The histogram for the long tail method from the JDOM benchmark is shown without outliers to preserve scale.

tions of a software artifact with other system elements often leads to behavior that is non-deterministic and, hence, best described using statistical models." The guide also lists an example course that covers our choice in general terms as "parameter estimation, statistical intervals, and statistical inferences."

The most detailed exposition is in the guidelines for computer engineering programs [7], which list "sampling distributions, estimation and hypothesis tests" among core topics. The guide again lists an example course that covers our choice as "calculation of mean and variance, central limit theorem, interval estimates and confidence intervals, testing hypotheses, t-test."

Another authoritative source of course information is the MIT Open Course Ware initiative [9]. Here, the most relevant course from the electrical engineering and computer science collection is called Statistics and Probabilistic Systems Analysis and Applied Probability. Again, the syllabus includes our choice of statistical methods in general terms, listing "the Law of Large Numbers, the Central Limit Theorem, estimation and hypothesis testing."

In the same source, a similar course from the mathematics collection is called Introduction to Probability. Again, the syllabus covers basic topics including "the Central Limit Theorem and frequentist significance tests and confidence intervals."

Finally, we can look at material covered in performance evaluation books. The classic "Art of Computer Systems Performance Analysis" by Jain [3] introduces basic summary statistics and confidence intervals; use of confidence intervals is explicitly preferred to hypothesis testing. Among more recent performance evaluation books, "Performance Evaluation and Benchmarking" edited by John and Eeckhout [4] follows similar trend.

3. EXPERIMENTAL EVALUATION

Both the confidence interval computation and the statistical hypothesis testing involve defining a probability of certain incorrect result – with confidence intervals, the confidence level defines the (complementary) probability of the interval missing the true value of the unknown metric; with hypothesis tests, the significance level defines the probability of the test rejecting a correct hypothesis. In practical experiments, it is important that the probability of such incorrect results is kept low – but this probability may actually differ from the chosen confidence or significance level because the computations are based on assumptions such as independent sampling, which are difficult to guarantee in reality.

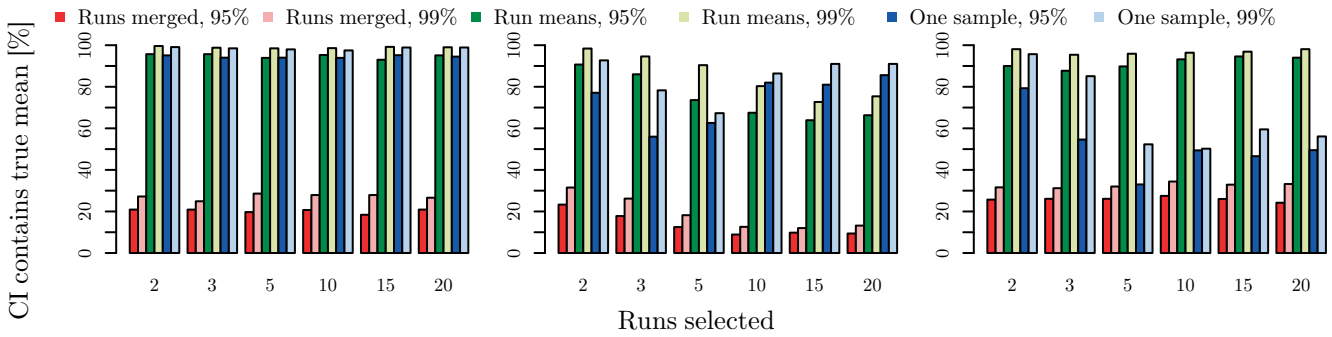
In our evaluation, we contrast the true probability of obtaining incorrect results against the chosen confidence or significance level. We use measurements from three data sets that represent different practical experiments:

- the maximum throughput metric collected from the SPECjbb2015 composite benchmark [12], running on an Intel Xeon E5-2620 V4 machine (8 cores at 2.1 GHz) with 64 GB RAM with Fedora Linux 25 and OpenJDK 1.8,
- the benchmark iteration times collected from the ScalaBench scaladoc large benchmark [11], running on Intel Xeon E5-2660 (32 cores at 2.2 GHz) with 48 GB RAM with Fedora Linux 24 and OpenJDK 1.8,
- the method execution times of two arbitrarily selected methods collected from unit testing benchmarks developed for the JDOM library [2], running on Intel Xeon E5-2660 (32 cores at 2.2 GHz) with 48 GB RAM with Fedora Linux 20 and OpenJDK 1.7.

Our experiments collect different measurements at different speeds. The SPECjbb2015 reports a throughput metric, one number per one run, which takes about 30 minutes in our configuration. The ScalaBench experiments report an execution time metric, one number per one iteration. After warm up, we perform 40 iterations in each run, which takes about 5 minutes in our configuration. The JDOM benchmarks report an execution time metric, produced continuously during execution. After warmup, we collect 1000 measurements in each run. For illustration, we show measurement histograms in Figure 1.

To collect sufficiently representative measurements, we run each benchmark multiple times. Except for the SPECjbb2015 experiments, which report one number per one execution, our data sets therefore have a large number of measurements coming from a smaller number of runs, and we have to decide how to apply the statistical computations to the measurements. Here, we examine three possible approaches:

- considering all measurements together. With no guidelines on how to treat measurements coming from multiple runs, one simple approach is ignoring the runs and treating the measurements together. On the plus side, the approach is simple and uses all data available. On the minus side, measurements from the same



(a) ScalaBench benchmark. (b) JDOM benchmark, long tail method. (c) JDOM benchmark, bimodal method.

Figure 2: How likely a confidence interval contains the true mean. The X axis shows how many runs are used to compute the confidence interval, the Y axis shows how likely it is that the interval contains the true mean.

run are more likely to be more statistically dependent than measurements from different runs, and this can influence the statistical computations. In the plots, we label this option "runs merged."

– considering entire runs rather than individual measurements. Another simple approach is aggregating all measurements in a run and using the aggregate statistic, for example sample mean per run, instead of the individual measurements. On the plus side, the approach is less likely to suffer from statistical dependency issues. On the minus side, the aggregation obviously loses information. In the plots, we label this option "run means."

– considering only one measurement from each run. On the plus side, the approach is least likely to influence the statistical computations with runs. On the minus side, again, the selection loses information, which is a problem when the run time is long. In the plots, we label this option "one sample."

3.1 Confidence Interval Computation

A common method of computing the confidence interval for sample mean relies on the fact that the probability distribution of sample mean is asymptotically normal. Here, we use the computation as provided by the `stats.t.interval` method of the SciPy package [10], for a more detailed explanation in the performance evaluation context see [1].

To provide a baseline to compare the confidence intervals against, we use an approach inspired by bootstrap – we collect many more measurements than we use to compute the confidence intervals, and compute many confidence intervals from random subsets of those measurements. We then compute the probability that such a confidence interval includes the sample mean of all measurements, which should approach the true confidence level achieved.

More formally, for a confidence interval from r runs at confidence level $1-\alpha$, when measurements are represented as a set of runs $M = \{R_1, R_2, \ldots, R_m\}$ and each run is represented as a set of samples $R_i = \{s_1^i, s_2^i, \ldots, s_n^i\}$, we compute as follows:

1. we set $\overline{M} = \sum_{i=1}^{m}\sum_{j=1}^{n} s_j^i/(m \cdot n)$ as the sample mean of all measurements,
2. we draw a random subset $M^{ci} \subset M$ with replacement such that $|M^{ci}| = r$ and compute the confidence interval ci from M^{ci},
3. we repeat step 2 multiple times and keep track of how many times $\overline{M} \in ci$,
4. using the count from above, we compute probability $P(\overline{M} \in ci)$; ideally, $P(\overline{M} \in ci) \approx 1-\alpha$.

Figure 2(a) illustrates that for the ScalaBench benchmark, the "runs merged" approach produces confidence intervals that miss the true mean with very high probability even for a high number of runs used – ideally, the probability should correspond to the confidence level used, as is roughly the case with the "run means" and "one sample" approaches.

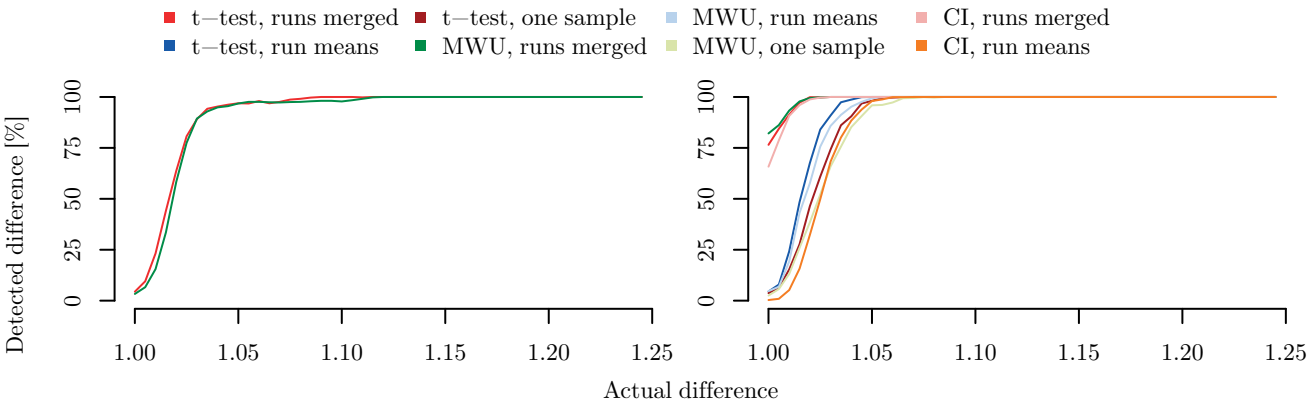
Figures 2(b) and 2(c) show the same results for the two methods from the JDOM benchmark, denoted as long tail method and bimodal method because of the shape of their probability distributions. We see that even the "run means" and "one sample" approaches may lead to confidence intervals that are surprisingly likely to miss the true mean.

We omit the results for the SPECjbb2015 benchmark, where the confidence interval computation works reasonably well.

3.2 Statistical Hypothesis Testing

Common statistical hypothesis tests used to detect difference in performance are Welch's t-test [13] and Mann-Whitney U-test [8]. The Welch's t-test assumes normally distributed data, however, it is considered robust enough to be used with other probability distributions; the null hypothesis postulates that the two compared distributions have equal means. The Mann-Whitney U-test does not assume any particular probability distribution; the null hypothesis postulates that a sample drawn from one of the two compared distributions is as likely to exceed a sample drawn from the other distribution as the other way around.

When the test assumptions are met, both tests should reject a true null hypothesis, or make a type I error, with probability equal to the significance level. To examine how much this holds for real data, we use the tests on many random subsets of measurements of the same benchmark. When comparing measurements of the same benchmark, the null hypothesis should hold by definition, the rejection rate would therefore ideally approach the significance level.



(a) SPECjbb2015 benchmark.

(b) ScalaDoc benchmark.

Figure 3: Difference detection sensitivity. The X axis shows the size (ratio) of the difference between the compared data sets, the Y axis shows how likely it is that a statistical test discovers the difference (rejects the null hypothesis). 5 runs per data set.

More formally, for a hypothesis test from r runs at significance level α, when measurements are represented as a set of runs $M = \{R_1, R_2, \ldots, R_m\}$ and each run is represented as a set of samples $R_i = \{s_1^i, s_2^i, \ldots, s_n^i\}$, we compute as follows:

1. we draw random subsets $M^a, M^b \subset M$ with replacement such that $|M^a| = |M^b| = r$ and evaluate the hypothesis test between M^a and M^b,
2. we repeat step 1 multiple times and keep track of how many times the hypothesis test rejected the null hypothesis,
3. using the count from above, we compute hypothesis rejection probability P; ideally, $P \approx 1 - \alpha$.

Apart from testing the type I error rate, we also look at the ability of each test to recognize a true difference between measurements. To do this, we introduce controlled difference between the compared distributions by taking random subsets of measurements of the same benchmark and multiplying one of the subsets by a constant. Then, we look at how much the rejection rate exceeds the type I error rate.

More formally, for a hypothesis test from r runs at significance level α with multiplication constant γ, when measurements are represented as a set of runs $M = \{R_1, R_2, \ldots, R_m\}$ and each run is represented as a set of samples $\{s_1^i, \ldots, s_n^i\}$, we compute as follows:

1. we draw random subsets $M^a, M^b \subset M$ with replacement such that $|M^a| = |M^b| = r$,
2. we set $M^c = \gamma \cdot M^b$ where multiplying a set with a scalar denotes multiplying all members of that set with the scalar,
3. we evaluate the hypothesis test between M^a and M^c,
4. we repeat steps 1 to 3 multiple times and keep track of how many times the hypothesis test rejected the null hypothesis,
5. using the count from above, we compute hypothesis rejection probability P.

To provide an additional reference, we include results for simple confidence interval overlap test alongside the Welsh's t-test and Mann-Whitney U-test results.

Figure 3(a) illustrates that for well behaved data sets, such as that of the SPECjbb2015 benchmark, the tests reliably detect a difference in means as small as 5 % from only 5 runs.

Figure 3(b) demonstrates how some tests fail with more measurements per run, even when the probability distribution itself is quite benign. Here, we use a total of 200 measurements from 5 runs to again reliably detect a difference in means as small as 5 %, but while the "run means" and "one sample" approaches ignore smaller differences, the "runs merged" approach reports even data sets where no difference was introduced as different around 70 % to 80 % of the time.

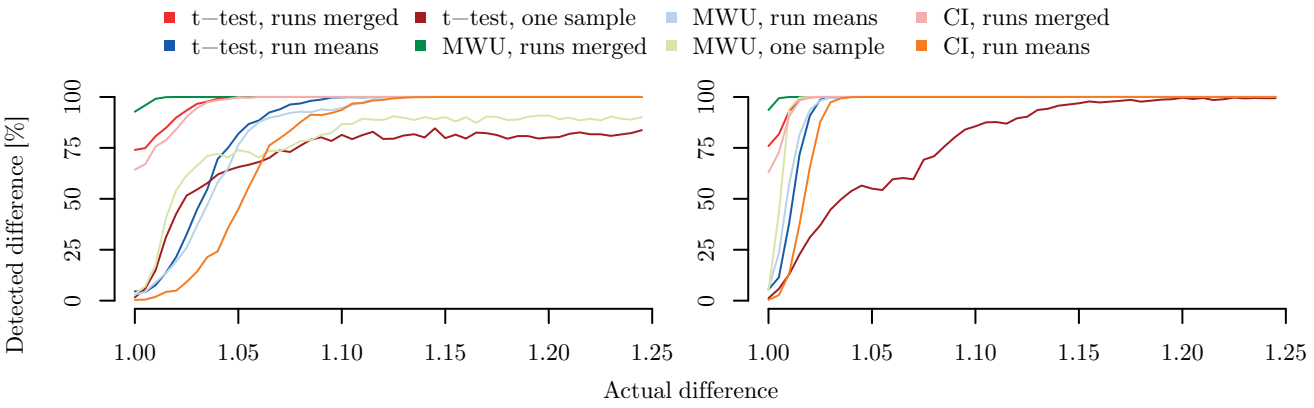
Finally, Figures 4(a) and 4(b) show that if the probability distribution of the measurements exhibits a long tail, the ability of the test to detect differences decreases dramatically. 5 runs, which constitute 5000 measurements, are only sufficient to reliably detect differences above 25 %, and 30 runs, which constitute 30000 measurements, are needed to again reliably detect a difference in means as small as 5 %.

Other combinations of benchmarks and run counts exhibit similar behavior and are omitted for brevity.

4. CONCLUSION

Our goal was to see how easily basic statistical methods apply to common practical measurements. We picked three common computations – confidence interval for the mean, Welch's t-test and Mann-Whitney U-test – and applied them to three data sets coming from a long running benchmark with small result variance (SPECjbb2015), a short running benchmark with multiple measurements per run (ScalaBench), and a short running benchmarks with high result variance (JDOM unit test, long tail method and bimodal method).

We have designed our experiments so that we can assess the quality of the computation results – the true confidence level for the confidence interval computation and the type I error rate and the change detection sensitivity for the hypothesis tests. The true value of these metrics is not known



(a) 5 runs per data set

(b) 30 runs per data set

Figure 4: Difference detection sensitivity. The X axis shows the size (ratio) of the difference between the compared data sets, the Y axis shows how likely it is that a statistical test discovers the difference (rejects the null hypothesis). JDOM benchmark, long tail method.

in common experiments, instead, an error rate corresponding with the computation parameters (confidence level, significance level) is intuitively expected.

We show that even in our rather simple configuration, the quality of the results very much depends on the properties of the measurements and details of how the computation is applied on the data. For the confidence intervals, we illustrate how a relatively straightforward application can lead to confidence intervals that miss the true mean an order of magnitude more often than the confidence level would suggest. For the hypothesis tests, we outline how certain approaches to application lead to an impractically high type I error rate, and how the sensitivity of the test very much depends on what kind of data the test examines.

The reasons for our results are not difficult to divine – they are mostly related to breaking some computation assumptions or relying on properties that hold only asymptotically. The point we raise is this: given what statistical analysis methods are taught in software related bachelor level courses, it may well be that the (poor) results we demonstrate are exactly the results our students will encounter. And while we may not have enough space to include more advanced methods, or even may not have methods that would work more reliably, we should carefully convey the practical limitations of the methods we do teach.

Acknowledgments

This work was partially supported by Charles University Institutional Funding (SVV) and by the Research Group of the Standard Performance Evaluation Corporation (SPEC).

5. REFERENCES

[1] A. Georges, D. Buytaert, and L. Eeckhout. Statistically Rigorous Java Performance Evaluation. In *Proceedings of the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications*, OOPSLA '07, pages 57–76, 2007.

[2] V. Horký, F. Haas, J. Kotrč, M. Lacina, and P. Tůma. Performance Regression Unit Testing: A Case Study. In M. S. Balsamo, W. J. Knottenbelt, and A. Marin, editors, *Computer Performance Engineering*, Lecture Notes in Computer Science, pages 149–163. Springer Berlin Heidelberg, 2013.

[3] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991.

[4] L. K. John and L. Eeckhout. *Performance Evaluation and Benchmarking*. CRC Press, 2005.

[5] Joint Task Force on Computing Curricula, ACM, and IEEE CS. Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, 2013.

[6] Joint Task Force on Computing Curricula, IEEE CS, and ACM. Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering.

[7] Joint Task Group on Computer Engineering Curricula, ACM, and IEEE CS. Computer Engineering Curricula 2016: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering, 2016.

[8] H. B. Mann and D. R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947.

[9] Massachusetts Institute of Technology. MIT OpenCourseWare. https://ocw.mit.edu/, 2017.

[10] SciPy team. SciPy: Scientific computing in Python. http://www.scipy.org/, 2017.

[11] A. Sewe, M. Mezini, A. Sarimbekov, and W. Binder. Da Capo Con Scala: Design and Analysis of a Scala Benchmark Suite for the Java Virtual Machine. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '11, pages 657–676. ACM, 2011.

[12] Standard Performance Evaluation Corporation (SPEC). SPECjbb 2015. http://www.spec.org/jbb2015/.

[13] B. L. Welch. The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.