Behavioral Log Analysis with Statistical Guarantees

Nimrod Busany
School of Computer Science
Tel Aviv University, Israel

Shahar Maoz
School of Computer Science
Tel Aviv University, Israel

ABSTRACT

Scalability is a major challenge for existing behavioral log analysis algorithms, which extract finite-state automaton models or temporal properties from logs generated by running systems. In this work we propose to address scalability using statistical tools. The key to our approach is to consider behavioral log analysis as a statistical experiment. Rather than analyzing the entire log, we suggest to analyze only a sample of traces from the log and, most importantly, provide means to compute statistical guarantees for the correctness of the analysis result. We present two example applications of our approach as well as initial evidence for its effectiveness.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Statistical methods*; D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Algorithms, Reliability

Keywords

Log analysis, specification mining

1. PROBLEM AND MOTIVATION

Running systems, be it web servers, virtual machines on the cloud, industrial robots, or network routers, generate logs that document their actions. The analysis of these logs carries much potential to improve software engineering tasks from documentation and comprehension to test generation and verification. Existing algorithms and tools for behavioral log analysis include various specification mining and model inference approaches, which extract finite-state automaton (FSA) models or temporal properties. Example approaches include [3, 10–12, 14, 15, 17, 21].

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
© 2015 ACM. 978-1-4503-3675-8/15/08...$15.00
http://dx.doi.org/10.1145/2786805.2803198

However, a major challenge for all behavioral log analysis algorithms is scale. Indeed, as logs become larger and consist of millions of lines, and as the semantics of a log line is stateful and depends on the lines that precede or follow it, directly or indirectly, when applied to real-world large logs, all existing algorithms take forever to complete and require unrealistic amount of memory.

In this work we propose to address the scalability of behavioral log analysis algorithms using statistical tools. Rather than analyzing the entire log, we suggest to analyze only a sample of traces from the log and, most importantly, provide means to compute statistical guarantees for the correctness of the analysis result.

Specifically, **we develop an approach to adding statistical guarantees to behavioral log analyses.** Given an analysis of interest and a sample of traces from a log, we compute the statistical confidence one may have in the analysis results. Conversely, given an analysis of interest and a required level of statistical confidence, we compute a stopping criteria, e.g., a sample size, for which indeed the analysis results can be trusted at the required statistical confidence level.

2. BACKGROUND AND RELATED WORK

While there has been much research on developing behavioral log analysis algorithms, only very little has been published on their scalability on the one hand and on the confidence one may have in their output on the other hand. Recently, Cohen and Maoz [6] have presented k-confidence, a confidence measure for k-Tails [4] which computes the probability that the log is complete (this work is extended to two other algorithms in [7]). These works do not provide any statistical guarantees. In contrast, our new work is based on *statistical hypothesis testing* and is thus much more robust and general than the one presented in [6, 7]; it can answer many other questions, beyond log completeness, as we later demonstrate in Section 3.3, and it provides the engineer with much flexibility in adapting to different algorithms and setting up required levels of accuracy on the one hand and statistical confidence on the other hand.

The approach we suggest is a pure black box approach. We do not look at the code of the program that created the log and take only the log itself as input. In many real-world situations, indeed the code is not available or is just too complex and too large to be a subject for static analysis.

It is important to note that our approach assumes that traces are randomly and independently sampled from the log and that the log adequately reflects the behavior of the system under investigation. This assumption may not al-

ways hold, e.g., if traces and logs depend on running tests that were generated according to some strategy. Our approach should therefore not be used as is as a means to evaluate the quality of test suites.

3. APPROACH AND UNIQUENESS

The key to our approach is to consider behavioral log analysis as a statistical experiment. For example, if the log is divided into traces, each new trace in the log is considered a trial which may contribute to the acceptance or rejection of an hypotheses about the system that generated the log. This setup allows us to apply well-known tools for statistical inference. We present basic definitions and show two examples.

3.1 Basic Definitions

A trace over an alphabet Σ is a finite word $\sigma = \langle e_1, e_2, \ldots, e_m \rangle$ where $e_1, \ldots, e_m \in \Sigma$. Let M be a model over an alphabet Σ. We use $T(M) \subseteq \Sigma^*$ to denote all traces accepted by the model M. A log of M, $l \subseteq T(M)$ is a finite set of traces from $T(M)$. We denote the set of all possible logs of M by $L(M)$.

In the behavioral log analysis setup, M and thus also $L(M)$ and $T(M)$ are unknown. All we know is a log l from $L(M)$. We assume that the distribution of traces in l closely resembles the distribution of traces in $T(M)$, i.e., that the log includes typical uses of the system under investigation.

3.2 Example I: k-Tails

k-Tails [4] is a well-known algorithm for extracting a candidate behavioral model from a log of execution traces. It has been extensively used in the dynamic specification mining literature and tools, in several variants, e.g., [1, 8, 16, 17, 19]. One variant of k-Tails, presented in [2], reads each trace and collects k-sequences, i.e., sequences of k consecutive events. It then uses these k-sequences to construct the k-Tails FSA. Most importantly, the FSA is uniquely defined by the set of k-sequences observed in the log.

Since a log may be too large to analyze, one would like to define a stopping criterion to indicate that "enough" traces were seen. For this purpose, we define a notion of δ-completeness as follows: a log $l \in L(M)$ is δ-complete when the total probability of all the unobserved k-sequences to be observed in the next trial (i.e., in the next randomly selected trace from $T(M)$), is smaller than or equals δ. δ is a statistical bound. Intuitively, in our context, δ can be viewed as a target sensitivity level to infrequent sequences.

Hypothesis testing as a stopping criteria. Our null hypothesis is that the log is not δ-complete, i.e., that the probability of a new random trace to reveal new information (include a sequence of events of length k that has not appeared in previously analyzed traces) is larger than δ. We stop reading new traces when this hypothesis can be rejected.

The experiment protocol proceeds iteratively as follows. At each step we pick a random trace from the log and check if the trace includes previously unobserved k-sequences; if so, we start a new experiment for the new knowledge base (i.e., the new set of all observed sequences so far); otherwise, we increment the number of trials since the last new k-sequence was observed; when this number is larger than N, the null hypothesis can be safely rejected. But where does N come from?

We model our analysis as a series of Binomial experiments [20]. Given a target sensitivity δ, and a statistical significance level α, we apply a Binomial proportion test [5] to compute N, the number of consecutive trials (new traces) that do not reveal new information required to reject the null hypothesis, i.e., to safely conclude that the log is δ-complete.

More formally, we describe the setup of iteratively randomly selecting a trace from a log and comparing the facts extracted from it (i.e., k-sequences in our example) against a knowledge base in terms of a Binomial experiment model [20] (this corresponds to a single experiment in the series):

- The experiment consists of N repeated independent trials (in our context, traces we have read since the last trace that revealed at least one new k-sequence)
- Each trial has a probability of success p (in our context, a success is a trace that reveals at least one new k-sequence)
- The probability of success does not change throughout the experiment (in our context, the knowledge base of k-sequences observed so far determines p)

Given the above modeling as a Binomial experiment, the distribution to observe k successful trials (i.e., traces revealing at least one new k-sequence) out of N trials is:

$$Bin(N,k) = \binom{N}{k} p^k (1-p)^{N-k}$$

Ideally, we would stop analyzing traces once all the k-sequences have been observed, i.e., when $p = 0$. However, in our context, we do not know the complete set of k-sequences and p is unknown. Therefore, we bound it inside a Binomial proportion confidence interval [5].

Definition 1 (Binomial proportion confidence interval)**.** Let $\hat{p}$ denote the proportion of successful trials over N random trials from $Bin(N,k)$; let z denote the $1 - \frac{1}{2}\alpha$ percentile of a standard normal distribution, where we refer to α as the error percentile. Then, p has a probability of $(1-\alpha)$ to be contained within the Binomial proportion confidence interval

$$[\hat{p} - z\sqrt{\frac{1}{N}\hat{p}(1-\hat{p})}, \hat{p} + z\sqrt{\frac{1}{N}\hat{p}(1-\hat{p})}]$$

Remark 1. The above formula relies on Normal approximation to a Binomial. We chose to present it due to its relative simplicity. However, it is inadequate when $p \approx 0$. Other, superior methods to compute a Binomial proportion confidence interval exist in the literature. In our calculations in the preliminary evaluation we used Jeffrey's interval [5]. The statistical mathematical details and the reason for our choice are outside the scope of this short paper.

Definition 2 ((α,δ)-confidence)**.** A (α,δ)-confidence reflects a $1-\alpha$ confidence that the true (but unknown) probability of success p is less than δ, i.e.,

$$\hat{p}_{upper\ bound(1-\alpha)} \leq \delta$$

Based on the Binomial proportion confidence interval formula, we find the stopping criteria by computing the number of unsuccessful consecutive trials N required to guarantee (α,δ)-confidence.

Remark 2. The proportion of success changes every time a new fact is learned. In k-Tails, when the knowledge base

is empty (i.e., no event sequence is known), the probability of success on any trace (longer than k) equals one. On the other extreme, when the knowledge base is complete (i.e., all possible event sequences of length k are known), the probability of success is zero. Therefore, when computing the proportion confidence interval, we must re-approximate $\hat{p}$ after learning a new fact, as the probability of success changes. This is why we start a new Binomial experiment after every success (i.e., after observing any new k-sequence).

We conclude with a concrete example. If we select a target sensitivity level of $\delta = 0.05$ and a significance level of $\alpha = 0.01$, Jeffrey's interval gives us $N = 77$. If we follow the protocol presented at the beginning of this section, we can stop reading traces from the log once we analyze $N = 77$ consecutive traces without new information. Then, we run k-Tails only on the sample of traces analyzed so far, and obtain $1 - \alpha = 0.99$ confidence level that the null hypothesis was correctly rejected, i.e., that the probability of any unobserved k-sequence to be included in the next (randomly selected) trace is less than $\delta = 0.05$. In short, we say that the sample is δ-complete with a significance level α.

Remark 3. Interestingly, and perhaps surprisingly, note that N depends on δ and α, but *not* on any specific detail of the k-Tails algorithm, not even the chosen k. However, this independency can be explained: the details of the k-Tails algorithm and the choice of the parameter k affect the very success or failure of each trial in the experiment. This points to a major advantage of our approach (also in contrast to [6,7]); it can be easily extended to any analysis algorithm that one can cast into the Binomial experiment protocol.

3.3 Example II: Property satisfaction frequency

Consider an engineer interested in the frequency of traces in the log that satisfy a certain property. For example, the property of interest may represent a bad scenario, and the engineer wants to estimate its frequency in a log coming from real-world executions of the system. As analyzing the entire log is infeasible, we are using statistical means to estimate this frequency. We sample traces, estimate the frequency, and would like to stop sampling once we know that the confidence interval around the estimated frequency is smaller than δ for a given statistical significance level α.

Let us refer to the true (but unknown) property satisfaction frequency in the system under investigation, i.e., in $T(M)$, by p, and the estimated frequency by $\hat{p}$. We define a notion of δ-similarity as follows: a log $l \in L(M)$ is δ-similar if the estimated frequency $\hat{p}$, computed as the proportion of satisfaction of the property in the traces in l, satisfies $|\hat{p} - p| \leq \delta$.

Given this notion, we propose the following iterative experiment protocol: pick a random trace from the log, check if the trace satisfies the desired property and update $\hat{p}$ to be the frequency of the property satisfaction observed in the traces so far; compute $d = |\hat{p}_{upper\ bound(1-\alpha)} - \hat{p}_{lower\ bound(1-\alpha)}|$; if d is smaller than δ stop sampling new traces. But where would the bounds come from?

Confidence interval as a stopping criteria. As done in the previous section, we define a Binomial experiment. Here, we change the interpretation given to a successful trial. A new trace is considered as a success, if it satisfies the desired property. The analysis of the confidence interval remains the same. In this context, the null hypothesis is that the true frequency p of the property lays outside of the confidence interval. We stop sampling (i.e., reject the null hypothesis), once we observe that the size of the confidence interval is sufficiently small. That is, when we stop, the log we have analyzed so far is δ-similar with a statistical significance level α. In other words, the probability that the true frequency p is far from $\hat{p}$ by more than δ is less than α.

As an example property of interest, consider an implementation of a caching mechanism on a web server. A log from such a server can be partitioned into traces based on user session and IP addresses. Then, an engineer may be interested in the frequency of a suboptimal scenario, where a user requests a page that she had already seen in the session but the server cannot fetch its contents from the cache.

Consider a concrete example. If we set $\alpha = 0.05$, after observing the property in 10 out of 50 traces (with a frequency $\hat{p} = 0.2$), we achieve 95% (1-α) confidence that the true frequency of the property, p, is in the interval $0.108 \leq p \leq 0.326$ ($d_1 = 0.218$). By reading more traces, we may be able to narrow this interval and get a more precise result, at the same significance level. For example, after observing the given property in 200 out of 1000 traces, for the same 95% (1-α) confidence we achieve $0.176 \leq p \leq 0.226$ ($d_2 = 0.05$). Thus, if we set $\delta = 0.05$ and use the procedure presented above, we would stop reading new traces at this stage.

Further, increasing the significance level α narrows the interval d as well. E.g., after observing the given property in 200 out of 1000 traces ($\hat{p} = 0.2$), with $\alpha = 0.10$ we get $0.18 \leq p \leq 0.221$ ($d_3 = 0.041$). This occurs since increasing α, increases the probability for an error (i.e., the probability that the true frequency lays outside of the interval).

Finally, an observation: $\hat{p}$ affects the interval size as well, e.g., with $\alpha = 0.05$, and after observing the given property in 50 out of 1000 traces, we get $0.038 \leq p \leq 0.065$ ($d_4 = 0.027$). The interval size d_4 is nearly half the size of d_2, obtained for the same sample size and the same significance level.

Remark 4. Note that the Binomial proportion interval size δ is monotonically decreasing in the significance level α and monotonically decreasing in the sample size N. This holds for any of the methods to compute the Binomial proportion interval. Also note that the δ is not symmetric around the estimated proportion $\hat{p}$. Therefore, to achieve δ-similarity, the interval size d must be less than δ (and not 2δ).

Remark 5. As in the previous example with k-Tails, here too, the stopping criteria depends on the user selected α and δ, but does *not* depend on the property of interest. The only assumption is that the analysis is able to decide, given a trace, whether it satisfies or does not satisfy the property. Thus, our approach is general and can be easily applied to many properties of interest.

4. PRELIMINARY RESULTS

As an early evaluation of our approach, we have implemented δ-completeness analysis for k-Tails with $k = 3$, and applied it to logs generated from four FSA models, taken from three previously published works [9, 13, 18]. The models varied in size and complexity: alphabet size ranged from 10 to 43 (mean 24.5), number of states from 6 to 18 (12.25), and number of transitions from 28 to 209 (88.75).

In our evaluation, we fixed the target sensitivity δ to 0.05. For three different statistical significance levels α, 0.15, 0.05, and 0.01, we computed the number of unsuccessful consecutive trials N required to guarantee (α,δ)-confidence: 31, 49,

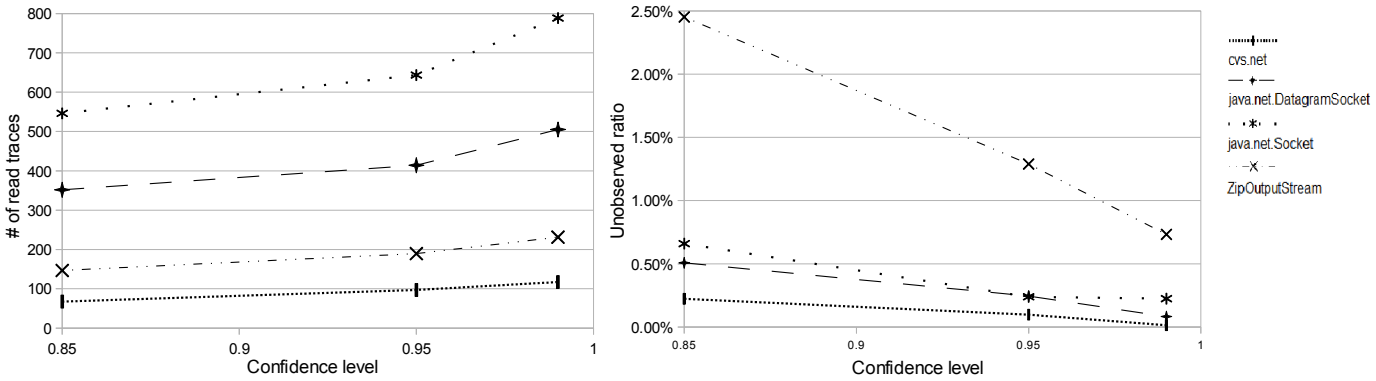

Figure 1: For δ fixed at 0.05: Confidence level (1-α) vs. # of read traces (left) and vs. unobserved ratio (right)

and 77 resp. For each of the four models, we used a trace generator to generate about 2000 random traces. We conducted the experiment according to the protocol described in Example I. We repeated each experiment 30 times for each of the four models and the three values of α.

In Fig. 1 we report the average number of traces analyzed (left) and the unobserved ratio (right), which is the ratio between the number of unobserved k-sequences and the total number of k-sequences possible in the model, achieved when reaching the stopping criteria.

As expected, to obtain lower statistical significance levels, we had to read more traces yielding lower unobserved ratio. Still, from the logs of 2000 traces we only had to analyze between 60 and 800 traces yielding unobserved ratio between 0.01% and 2.45%. The variance in number of traces and in the unobserved ratio can be attributed to the variance in the four models size and complexity.

Note that we do not report the ratio between the original log size and the number of traces we have read, as the stopping criteria is independent of the original log size. Indeed, one should expect similar number of read traces for logs of any size, as large as we want. Thus, the potential gain in scalability is unbounded.

δ-completeness was achieved in all experiments for all models, which hints that our analysis is conservative and can be further improved to reduce the number of read traces.

5. CONTRIBUTIONS AND FUTURE WORK

In this short paper we introduced the use of trace sampling and statistical inference to address the scalability challenge in the behavioral analysis of large logs. The key to the approach is to consider each new trace in a log as a trial in an experiment. We demonstrated the application of our approach to two different analyses: the k-Tails specification mining algorithm and the problem of computing the frequency of a property satisfaction in the log. Preliminary evaluation provides evidence for the reliability of our calculations and for the usefulness of our approach in dealing with large logs.

We are working on the following future research. First, we investigate other, more elaborated stopping criteria. Second, we extend our evaluation beyond the four models mentioned above to real-world logs provided by our industrial partners. Finally, we look for additional behavioral log analyses where our statistical inference approach can be applied.

6. REFERENCES

[1] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining API patterns as partial orders from source code: from usage scenarios to specifications. In *ESEC/SIGSOFT FSE*, pages 25–34, 2007.

[2] I. Beschastnikh, Y. Brun, J. Abrahamson, M. D. Ernst, and A. Krishnamurthy. Unifying FSM-inference algorithms through declarative specification. In *ICSE*, pages 252–261, 2013.

[3] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *SIGSOFT FSE*, pages 267–277, 2011.

[4] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Comput.*, 21(6):592–597, June 1972.

[5] L. Brown, T. Cai, and A. DasGupta. Interval Estimation for a Binomial Proportion (with discussion). *Statistical Science*, 16(2):101–133, 2001.

[6] H. Cohen and S. Maoz. The confidence in our k-tails. In *ASE*, 2014.

[7] H. Cohen and S. Maoz. Have we seen enough traces? In *ASE*, 2015. To appear.

[8] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998.

[9] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S. Hack, and A. Zeller. Automatically generating test cases for specification mining. *IEEE Trans. Software Eng.*, 38(2):243–257, 2012.

[10] F. C. de Sousa, N. C. Mendonça, S. Uchitel, and J. Kramer. Detecting implied scenarios from execution traces. In *WCRE*, pages 50–59, 2007.

[11] D. Fahland, D. Lo, and S. Maoz. Mining branching-time scenarios. In *ASE*, pages 443–453, 2013.

[12] S. Kumar, S.-C. Khoo, A. Roychoudhury, and D. Lo. Mining message sequence graphs. In *ICSE*, pages 91–100, 2011.

[13] D. Lo and S.-C. Khoo. QUARK: Empirical assessment of automaton-based specification miners. In *WCRE*, 2006.

[14] D. Lo and S. Maoz. Scenario-based and value-based specification mining: better together. In *ASE*, 2010.

[15] D. Lo, S. Maoz, and S.-C. Khoo. Mining modal scenario-based specifications from execution traces of reactive systems. In *ASE*, pages 465–468, 2007.

[16] D. Lo, L. Mariani, and M. Santoro. Learning extended FSA from software: An empirical assessment. *Journal of Systems and Software*, 85(9):2063–2076, 2012.

[17] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *ICSE*, pages 501–510, 2008.

[18] M. Pradel, P. Bichsel, and T. R. Gross. A framework for the evaluation of specification miners based on finite state machines. In *ICSM*, pages 1–10, 2010.

[19] S. P. Reiss and M. Renieris. Encoding program executions. In *ICSE*, pages 221–230, 2001.

[20] S. M. Ross. *Simulation, Fourth Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[21] N. Walkinshaw and K. Bogdanov. Inferring finite-state models with temporal constraints. In *ASE*, pages 248–257, 2008.