2016 IEEE/ACM 38th IEEE International Conference on Software Engineering

Behavioral Log Analysis with Statistical Guarantees

Nimrod Busany 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 paper we present *statistical log analysis*, which addresses 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 the theoretical foundations of our approach and describe two example applications, to the classic k-Tails algorithm and to the recently presented BEAR algorithm.

Finally, based on experiments with logs generated from real-world models and with real-world logs provided to us by our industrial partners, we present extensive evidence for the need for scalable log analysis and for the effectiveness of statistical log analysis.

CCS Concepts

•**Software and its engineering** → **Dynamic analysis;** *Software verification;*

Keywords

Log analysis, specification mining

1. INTRODUCTION

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, 11, 15, 18, 22, 23, 25, 34].

However, a major challenge for all behavioral log analysis algorithms is scale. Indeed, recent work has reported that existing implementations of several well-known log analysis algorithms run out of memory or take hours to complete when executed over logs of several hundred MBs [35].

In this work we propose *statistical log analysis*, which addresses scalability of behavioral log analysis using statistical tools. Rather than analyzing the entire log, we suggest to analyze only a sample of entries 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 entries 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 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.

Following the presentation of our general approach, we demonstrate it by concretely applying it to two different previously published analyses. First, we apply statistical log analysis to the well-known k-Tails algorithm [4], which extracts a candidate behavioral model from a log of execution traces, based on the set of sequences of k consecutive events found in the log (we use the variant presented by Beschastnikh et al. in [2]). Second, we apply statistical log analysis to the BEAR algorithm, recently presented by Ghezzi et al. in [17], which builds a set of Discrete Time Markov Chains (DTMC) of users' navigational behaviors recorded in logs, based on the frequencies of transitions found in the log.

We have implemented our approach and evaluated it on three sets of logs. Our experiments show that as the size of the logs grow, their analysis becomes challenging, and that statistical log analysis can be used to significantly reduce the computation effort of the analysis while still providing highly reliable results. We provide a detailed account of the evaluation we have performed in Sect. 4.

It is important to note that the scalability we aim for in statistical log analysis is sub-linear: the complexity of the analysis we do is less than linear to the size of the input log. Note that this is a rather strong notion of scalability. In fact, our experiments show that as the log size increases, the size of the sample we analyze approaches a constant! Indeed, the stopping criteria we compute depend on the confidence one

ICSE '16, May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2884781.2884805

may want in the correctness of the results, but are independent of the log size. As real-world log sizes are unbounded, this strong notion of scalability is indeed required.

We further note that our present work does not make any new claims regarding the usefulness of the results of behavioral log analyses such as k-Tails and BEAR. Based on claims and evidence provided in previous works, by others, as cited above, we assume that these analyses' results could be useful for software engineers, and focus in the present work solely on the challenge of scalability and on the statistical means we propose to use in order to address it.

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.

Finally, we note that our approach assumes that relevant entries (e.g., 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 always 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.

Paper organization. We start off with a short discussion of related work. Sect. 3 presents an overview of our approach and its unique features, using two concrete example applications. Sect. 4 presents an extensive evaluation over three sets of logs. Finally, Sect. 5 concludes with a summary of contributions and future work directions.

2. RELATED WORK

While there has been much research on developing behavioral log analysis algorithms (see, e.g., [2,3,11,13–19,22, 23, 25, 27, 28, 30, 31, 34, 36]), only very little has been published on their scalability and on the confidence one may have in their output. Recently, Cohen and Maoz [7] 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 [8]). These works do not provide statistical guarantees. In contrast, our work is based on statistical log analysis and is thus much more robust and general than the one presented in [7,8]; it can answer many other questions, beyond log completeness, as we later demonstrate, and provides engineers 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.

One means to achieve scalability in log analysis is distributed computing. In [35], the authors cast several different specification mining algorithms to the MapReduce model. In [26], the authors present the use of MapReduce to scale k-Tails. We consider distribution to be an orthogonal approach to ours. We note that a distributed computing framework is not always available or easy to use. Moreover, the scale-up in running time is in theory at most linear in the number of computing resources, and in practice even less; in the results reported in both works [26,35], the marginal contribution of adding more resources quickly decreases.

Finally, in a recent FSE NIER track short paper [6], we presented an overview of our approach with preliminary evaluation. Our present paper extends [6] significantly, with a comprehensive presentation of our approach and an extensive evaluation performed over real-world data.

3. STATISTICAL LOG ANALYSIS

The key to statistical log analysis is to consider behavioral log analysis as a statistical experiment. For example, if the log is partitioned 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. The granularity of the partition is up to the user to define, based on the specific analysis of interest. The key property for the selection of partition granularity is that its parts (e.g., traces) can be independently sampled from the log.

To cast the behavioral analysis into a statistical experiment, we assume that the log is a good representation of the system generating it. Further, we assume that the system and its environment do not change over time, to exclude influence of external biases on the traces. In stochastic terms, we assume that the system corresponds to a time-homogeneous process. The traces are assumed to be generated identically and independently from the same underlying distribution. Finally, our approach as presented here can only be applied to incremental inference algorithms, where each trace can be processed independently and its contribution can be determined with respect to the current state of the algorithm.

Still this most general setup allows us to apply well-known tools for statistical inference. We present basic definitions and show two example concrete applications.

3.1 Basic Definitions

A trace over an events 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 S be a system generating traces over an alphabet Σ. We denote by $D(S) : \Sigma^* \to [0, 1]$, the discrete probability distribution of traces when executing the system in its environment. A log of S, denoted by l, is a finite set of traces from $D(S)$. In the behavioral log analysis setup, S and $D(S)$ are unknown. All we know is a log l from $D(S)$. We assume that the distribution of traces in l closely resembles the distribution of traces in $D(S)$, i.e., that the log includes typical uses of the system under investigation. We have no information about the system S other than the log l.

Filters may be applied to the log l, so as to remove events that should not be considered by the analysis. Without loss of generality, we further assume that the log l (possibly after filtering) is partitioned according to some user-defined criteria, such that each event that appears in l belongs to exactly one part of the partition. An example partition is one that is based on traces. Another example is a partition where each part consists of exactly one event.

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, 9, 24, 25, 32]. 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-similarity as follows: a log l is δ-similar 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 $D(S)$), is smaller than or equals δ. δ is a statistical bound. Intuitively, in our context, δ can be viewed as a target insensitivity level to infrequent sequences.

Hypothesis testing as a stopping criteria. Our null hypothesis is that the log is not δ-similar, i.e., that the probability of a new random trace to reveal new information (include a k-sequence 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 is iterative. 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 [33]. Given a target insensitivity 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-similar.

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 [33] (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 s successful trials (i.e., traces revealing at least one new k-sequence) out of N trials is:

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

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 $Bin(N,s)$; 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

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

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 evaluation (Sect. 4), we used Wilson interval recommended by [5], which provides high coverage probability for a broad range of p values. We note that for $p \approx 0$ and $p \approx 1$, Agresti-Coull interval may be more appropriate. Lastly, we note that Jeffreys prior interval is less conservative and tends to yield coverage probability closer to $1-\alpha$.

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 k-sequences 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. For this reason, we start a new Binomial experiment after every success (i.e., after observing a trace with a new k-sequence).

We conclude with a concrete example. If we select a target insensitivity level of $\delta = 0.05$ and a significance level of $\alpha = 0.01$, Willson's interval gives us $N = 130$. If we follow the protocol presented above, we can stop reading traces from the log once we analyze $N = 130$ consecutive traces without new information. The statistical guarantees are about the completeness of the obtained set of k-sequences. When the analysis terminates and the null hypothesis is rejected, we have a $1-\alpha = 0.99$ confidence that the total probability of any of the unobserved k-sequences to appear in a random trace, denoted by p, is less than $\delta = 0.05$. In short, we say that the sample is δ-similar with a significance level α.

Remark 3. Interestingly, and perhaps surprisingly, note that N depends on $\hat{p}$, δ, and α, but *not* on any specific detail of the k-Tails algorithm, not even the chosen k. However, this independence 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 [7,8]); it can be easily extended to any analysis algorithm that one can cast into the Binomial experiment protocol.

Remark 4. Note that while reading additional data is expected to increase the reliability of the resulting model, by definition of the stopping criteria, it is redundant (and a waste of resources) in terms of the required statistical bound.

3.3 Example II: BEAR inference algorithm

BEAR [17] is a recently presented inference algorithm that constructs a set of Discrete Time Markov Chains (DTMC) of users' navigational behaviors recorded in logs. The inferred models are then analyzed in order to verify quantitative properties by means of probabilistic model checking. The algorithm constructs DTMCs according to user classes. Each user is assigned to one or more of these classes, and her user sessions (i.e., traces) are used in the construction of the corresponding DTMC. DTMCs are constructed according to transitions' frequencies in the log from which they are mined. Most importantly, the set of transitions' frequencies found in the log determines the DTMC model that will be built.

Consider an engineer applying BEAR over a log from a particular user class. Running BEAR over the entire log may be very slow, therefore we apply our statistical log analysis to the problem. Rather than reading the entire log, we sample events from it. Each sampled event is a trial in an experiment. A transition is created by connecting each event in a user session to its preceding event. We estimate the transitions' frequencies, and would like to stop sampling once we know that the confidence interval around the estimated frequencies is smaller than δ for a given statistical significance level α.

We describe the procedure for a single transition, then simply apply it over the set of all observed transitions. We refer to the true (but unknown) transition frequency in the system under investigation, i.e., in $D(S)$, by p, and the estimated frequency by $\hat{p}$. We define a notion of δ-similarity as follows: a log l is δ-similar if the estimated frequency $\hat{p}$, computed as the proportion of transitions which equal to the desired transition 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 and traverse its transitions. If a transition equals to the transition at hand, increase the transition occurrence counter and the total transitions occurrence counter, and update its frequency $\hat{p}$. Otherwise, if a transition is not equal to the transition at hand, increase the total transitions occurrence counter, and update its frequency $\hat{p}$; compute $d = |\hat{p}_{upper\ bound(1-\alpha)} - \hat{p}_{lower\ bound(1-\alpha)}|$; if d is smaller than δ stop sampling. 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 of a successful trial. First, a new trial is defined every time a transition is read. Second, a new transition is considered a success, if it equals to the transition at hand. The analysis of the confidence interval remains the same. In this context, the null hypothesis is that the true frequency p of the transition 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 of a transition, p, is far from $\hat{p}$ by more than δ, is less than α.

As an example for a transition of interest, consider web application of a hotel. A log from such an application can be partitioned into traces based on IP addresses and a time window. Now, consider a user browsing through the catalog of rooms, selecting a room and reaching the make-a-reservation page. Then, instead of making a reservation, the user hits the back button. An engineer may be interested in the frequency of users who take this transition, in order to make an informed decision regarding a possible change in the make-a-reservation page.

Consider a concrete example. If we set $\alpha = 0.05$, after observing a transition in 10 out of 50 transitions (with a frequency $\hat{p} = 0.2$), we achieve 95% (1-α) confidence that the true frequency of the transition, p, is in the interval $0.112 \leq p \leq 0.33$ ($d_1 = 0.218$). By reading more user sessions, we may be able to narrow this interval and get a more precise result, at the same significance level. For example, after observing the desired transition in 200 out of 1000 transitions, 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, reducing the significance level (1-α) narrows the interval d as well; e.g., after observing the desired transition in 200 out of 1000 transitions ($\hat{p} = 0.2$), with $\alpha = 0.10$ we get $0.18 \leq p \leq 0.222$ ($d_3 = 0.042$). 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 desired transition in 50 out of 1000 transitions, 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 significance level.

In this experiment, the statistical guarantees are about the accuracy of the obtained frequency. When the analysis terminates, we have 1-α confidence that the distance between the true and obtained frequencies is smaller than δ.

Remark 5. As in the k-Tails example, here too, the stopping criteria depends on $\hat{p}$, δ, and α, but does *not* depend on the transition of interest. Note that the interval size δ is monotonically decreasing in the significance level α and monotonically decreasing in the sample size N. We note that the δ may not be symmetric around the estimated proportion $\hat{p}$ (depnding on the selected interval). Therefore, to achieve δ-similarity, the interval size d must be less than δ.

Remark 6. Note that our key assumption is that the analysis is able to decide, given a transition, whether it equals to the transition at hand. Thus, the example application to the BEAR algorithm, which estimates the frequencies of transitions, is actually an instance of a more general approach to estimate the frequency of satisfaction of any property of interest, given that satisfaction in a sampled part (e.g., a trace) can be decided. In this experiment, multiple independent trials are performed. In each we check for the satisfaction of a property of interest. A trial is considered a success if the sampled part satisfies the property of interest and a failure otherwise. Calculating bounds for property satisfaction frequency is done similarly to the way we have done it for BEAR in the special case of transitions.

Remark 7. One may notice that the trials in the procedure above are not entirely independent (violating an important assumption). In fact, transitions in the same part, i.e., a user session, are in most cases highly correlated. Nevertheless, our method is applied over large logs, containing many parts, whose length is usually significantly smaller than the log size. Furthermore, parts are randomly selected, which eliminates correlation between events from different parts. Therefore,

the correlation between events in similar sessions may not have a significant effect on the analysis. We empirically test the soundness of our approach in Sect. 4.

4. EVALUATION

The research questions guiding our evaluation are:

RQ1 As real-world logs become larger, will existing log analysis algorithms scale?

RQ2 Does our approach reach a required high reliability (soundness)?

RQ3 Does our approach allow one to reduce the number of traces read while maintaining high reliability?

RQ4 What is the overhead of computation time in our approach and does it actually allow scalable sub-linear execution times?

4.1 Logs Used in Evaluation

In the evaluation we have used three sets of logs.

4.1.1 Log Set I - Logs generated from FSA models

We conducted the *k-Tails experiment* (Sect. 3.2) over logs that we have generated from four FSA models of real systems, taken from three previously published works [10, 20, 29]. The models varied in size and complexity: alphabet size ranged from 10 to 42 (mean 24.5), number of states from 6 to 18 (12.25), and number of transitions from 28 to 209 (88.75). To generate the logs from the models we used the trace generator from [24], with path coverage. We fixed the number of traces in all logs to 25K; the number of events ranged from about 252K to about 605K, and the average trace length from 7.37 to 24.21. To parse the above logs, we used the parser from the implementation of Synoptic [3].

4.1.2 Log Set II - Daily regression logs from a telecommunications company

We repeated the *k-Tails experiment* (Sect. 3.2) with a set of five real-world logs, which we received from our partner, a team in a large multi-national telecommunications equipment company. The set includes logs of high-level API calls, system events, distribution of tasks, DB queries, and communication between processes over different machines. All logs were generated by a single run of a daily regression, which was executed in debug mode. Again to parse the logs we used Synoptic's parser cited above. As the parser requires the user to specify regular expressions, we have defined these for each of the five logs, after consulting with our colleagues at the company. The logs varied in size and complexity: alphabet size ranged from 10 to 115, number of events from 10.7K to 40K, number of traces from 194 to 1261, and average trace length from 13.8 to 55.31.

The number of traces contained in the logs above was rather small (as they only contained a single day of regression run). Since gaining statistical confidence requires large logs, we decided to duplicate the original files, by duplicating each entry multiple times, thus creating multiple copies of the same day. To duplicate the logs, we first read the entire log, then duplicated indexes, which we mapped to the parsed traces. We duplicated each index multiple times, shuffled them, popped one index at a time, and read its corresponding trace. The advantage of duplicating the original logs is that it enables us to fix the amount of information that the log contains, while increasing its size. This allows us to better interpret the experiments' results.

4.1.3 Log Set III - Real-Estate REST application logs

To conduct the BEAR algorithm experiment (Sect. 3.3), we used BEAR's dataset, generously given to us by the authors of [17]. The set consists of a single log, which was generated by a Real-Estate REST application, executed by an IT consultancy company.

The log contains rows which record requests of web resources issued by clients to the application's Web server. Each request is considered to be an event. To parse the log we used the code and filters (expressed by regular expressions) provided by the authors of [17]. The log consists of about 390K events, collected over two years. The alphabet size is 29. We followed [17] and filtered out events (i.e., requests) that correspond to secondary resources (e.g., requests for CSS or Javascript resources), as they do not represent users' navigational behaviors. After applying the filters, about 39.5K events are used in the construction of the DTMCs.We partitioned the events in the log into users' sessions based on IP addresses and a user window of 100 minutes. Events with similar IP, separated by a longer time period were assigned to different traces (i.e., user sessions). This filtering and partitioning are consistent with the ones chosen by the authors of [17].

Finally, since almost all of the events originate from users of the Mozilla Firefox browser (about 38K events), we focused on constructing a model for this user class. This yielded a single DTMC, which captures these users' interactions with the application. We filtered out events that did not belong to this user class.

For more details about the log and filters used, we refer the reader to the original paper about the BEAR approach [17].

4.2 Setup and Methodology

We executed the experiments on an ordinary laptop computer, Dell XPS 15, Intel Core i7-4712HQ CPU@2.3GHz, 16GB RAM, Samsung SSD PM851 mSATA 512GB.

4.2.1 Binomial confidence interval calculator

We implemented the Binomial confidence interval calculator used in the evaluation with the `ConfidenceInterval` class included in Apache Commons Math 3.3.

4.2.2 k-Tails experiment

To implement the *k-Tails experiment* (Sect. 3.2), we modified the code of Invarimint [2] to support the use of the Binomial confidence interval calculator when analyzing traces.

To apply our approach with k-Tails, we first read the entire log and shuffled the traces. We then run the k-Tails algorithm with Invarimint's implementation [2], which we modified to identify if new information was revealed after a new trace is read. Then, we performed (α, δ)-confidence check after reading each trace with the Binomial confidence interval calculator. We ended the experiment once the desired level of confidence was obtained. The k-Tails parameters we used were $k = 1, 2, 3$.

4.2.3 BEAR algorithm experiment

In this experiment we first selected a set of transitions whose frequencies' we want to bound. We decided to consider any transition observed as we process the log as an ex-

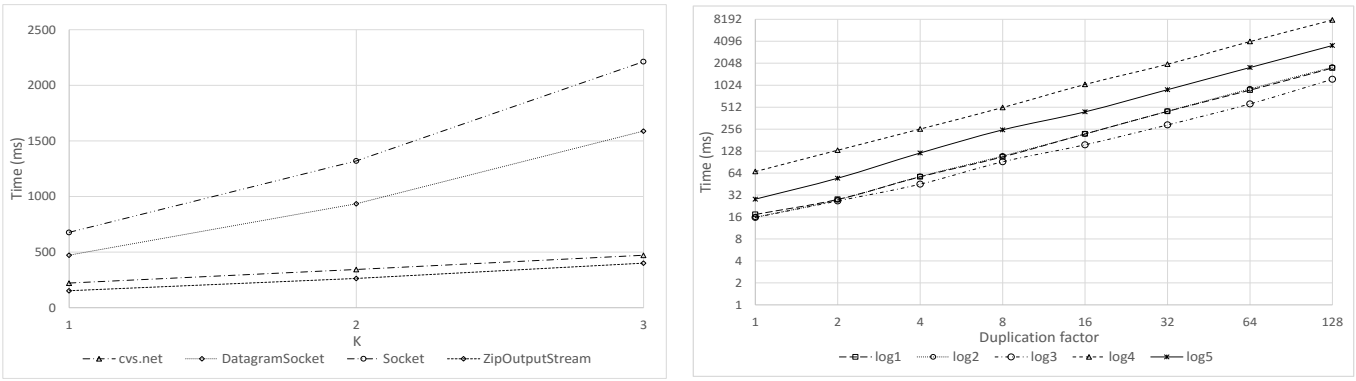

Figure 1: Running k-Tails over Log Set I (left) and Log Set II (right), see Section 4.3.1

.

periment, i.e., once a new transition is observed, it initiates a new experiment that is added to the set of running experiments. We stopped processing the log once all the running experiments had achieved the desired (α, δ)-confidence. To test how reliable is the stopping criterion, we stop updating the frequency of transitions whose experiment has ended. In other words, we avoid updating transitions for which we already achieved a confidence interval smaller than δ.

To implement, we modified BEAR's implementation to support the use of the Binomial confidence intervals. First, for the Binomial confidence interval to be correctly computed, user interaction sessions must be randomly selected from the log. To cope with this, we modified BEAR's parser to first split the log into user sessions according to IP and a user window. Then, at each iteration of the algorithm, we select a user session and pop its events sequentially. After we read all the user session events, we remove it and randomly select a new user session.

When selecting an event, we use the BEAR algorithm to construct a transition that connects it to the preceding event in the user session (initial events are connected with a dummy 'start' node). Then, we update the transitions' frequencies for all of the active experiments accordingly, as explained in Sect. 3.3. Further, if the transition is observed for the first time, we initiate a new experiment for it. To do this, we initialize a new Binomial confidence intervals calculator setting its initial trials count to the number of read events and its number of successes to one. [1]

Finally, we compute the confidence intervals around all transitions with active experiments and terminate any experiment with a calculated interval of size smaller than δ. We stop reading events once all the experiments are terminated (i.e., when the desired (α, δ)-confidence level is obtained for all transitions), or when the entire log is read.

4.2.4 Measures

The measures we use to evaluate the results of our experiments are similarity, reliability, absolute and ratio of sample size, and absolute and ratio of analysis execution time.

[1]We note that an experiment is not initiated for a new transition after we obtained a tight confidence interval for non-observed transitions. For example, if we already saw 450 transitions, then the confidence interval of any unobserved transition is between 0 to 0.008, hence smaller than δ, so we treat it as a completed experiment.

Similarity. We define *similarity* as the distance between the true parameter that the experiment estimates and its estimated value when the experiment is completed.

In the k-Tails experiment, the parameter used in the *Hypothesis testing as stopping criteria* experiment is the probability p of a new random trace to reveal new information (see Sect. 3.2). Since we do not know p, we use the *Maximum Likelihood Estimator (MLE)* [12] of p over the entire log, denoted by p_{MLE}. We compute it by taking the ratio between traces containing k-sequences missing from the partial log, and the total number of traces in the entire log. Then, *similarity* is defined as $1 - |p_{MLE} - \hat{p}|$. Since the k-Tails experiment only ends when $\hat{p} = 0$, the similarity is simply $1 - |p_{MLE}|$, and can also be interpreted as a measure of completeness.

In the BEAR algorithm experiment, the property used in the *Confidence interval as stopping criteria* experiment is the true frequency of a transition, denoted by p. Since we do not know p, we compute the *MLE* of p, denoted by p_{MLE}, which is the frequency of the transition in the entire log. Then, we compute the distance between the transition's frequencies in the partial log, denoted by $\hat{p}$, and p_{MLE}. Again, *similarity* is defined as $1 - |p_{MLE} - \hat{p}|$. Intuitively, when the two are identical, *similarity* equals 1, and when the two are most distant, *similarity* equals 0.

δ-Reliability (precision). We define *δ-reliability* as the experiment's precision, where a true positive (false positive) is the case where the experiment was correctly (incorrectly) terminated.

In the k-Tails experiment, the null hypothesis is that the true probability of a new random trace to reveal new information (i.e., a new k-sequence), p, is greater than δ. The experiment terminates once we have (α, δ)-confidence that p is smaller than δ. Since we use the p_{MLE} of p, we declare the experiment successful if $p_{MLE} \leq \delta$. Since $1 - \alpha$ captures the theoretical reliability of the experiment (assuming the underlying assumptions hold, see Sect. 3), we expect the success rate, i.e., the δ-reliability of a set of experiments, to be $1 - \alpha$.

In the BEAR algorithm experiment, we stop once we gain $1 - \alpha$ confidence that the estimated frequency $\hat{p}$ of a transition and the true frequency of a transition p are within a distance of δ. Since we use the p_{MLE} of p, we declare the experiment successful if $|p_{MLE} - \hat{p}| \leq \delta$. Again, since

$1-\alpha$ captures the theoretical reliability of the experiment, we expect the δ-reliability of a set of experiments to be $1-\alpha$.

Absolute and ratio sample size. The absolute sample size is the number of trials performed (traces or transitions analyzed) in the experiment. The ratio sample size is the ratio between the absolute sample size and the size of the entire log (number of traces or transitions).

Absolute and ratio of analysis execution time. The absolute sample analysis execution time is the time required to analyze the sample (e.g., to extract k-sequences). The ratio of analysis execution time is the ratio between the absolute sample analysis execution time and the time required to analyze the entire log.

4.3 Results

4.3.1 RQ1

To answer RQ1, we conducted the following experiments. We run k-Tails over Log Set I and computed the execution time of extracting k-sequences from traces. Execution times as function of k are presented in Fig. 1 (left). These results illustrate that the complexity of analysis (i.e., the chosen k) can greatly affect the execution time of mining k-sequences. Furthermore, the richness of the log is a prominent factor. As an example, one may observe the large gap between the results for the `DatagramSocket` log and the `ZipOutputStream` log. Indeed, the `DatagramSocket` model contains a larger alphabet and more transitions than the `ZipOutputStream` model. As a result, its traces tend to be longer and so is the log size. Moreover, while the number of events in the `DatagramSocket` log is 2.44 times larger, the computation time it requires is 4.43, 5, and 5.51 times longer than that of the `ZipOutputStream` log for k=1, 2, 3 resp.

We run k-Tails over Log Set II. Since these logs were rather small, we also run k-Tails over the duplicated logs. Fig. 1 (right) shows the execution times for duplication factors $1, 2, 4, \ldots, 128$. These experiments illustrate a linear increase in the execution time as the logs increase in size, which shows that for truly large logs, even a simple collection of k-sequences may not be feasible.

Finally, we run the original implementation of BEAR over Log Set III and measured the execution time. We repeated it 10 times. The average time measured was 12 minutes.

To answer RQ1, we see that the size and complexity of logs and the complexity of the analysis algorithms can greatly affect the analysis computation time. As the size of the logs grow, their analysis becomes challenging.

4.3.2 RQ2

To answer RQ2, we conducted the following experiments. First, we run the k-Tails experiment over Log Set I and Log Set II. For Log Set I, we fixed the target insensitivity δ at 0.05, and used three different statistical significance levels α, 0.01, 0.05, and 0.15. We invoked the Binomial interval confidence calculator after reading each trace. The numbers of unsuccessful consecutive trials N required to guarantee (α,δ)-confidence are 130, 73, and 40 resp. We repeated each experiment 30 times for each of the four logs and the three values of α and k.

Fig. 2 (left) [2] reports the similarity levels obtained when reaching the stopping criteria. As can be seen, the values of the first quartile for $\alpha = 0.01, 0.05$ in all models are higher than 95%. This shows that in the large majority of experiemnts the desired similarity level was indeed obtained with these significance levels. Furthermore, 92.01% of all experiments obtained an average similarity level of 90% and higher. This illustrates reliability for Log Set I.

Interestingly, in deeper analysis of results not shown in the figure, we observed that the average δ-reliability(0.05) levels for $\alpha = 0.01$ are 100% for all models. Moreover, when setting $\alpha = 0.05$, the δ-reliability(0.05) levels for the four logs were 98.8%, 91.6%, 100%, and 92.2%. For $\alpha = 0.15$, the δ-reliability(0.05) levels for the four logs were 90%, 95.5%, 54.4%, and 31.1%. This shows that for low values of α the intervals tend to be too conservative, while for high values of α, the intervals may be not conservative enough.

For Log Set II we conducted these experiments in a similar way (experimenting with similar parameters). Fig. 2 (right), reports the average similarity levels obtained when reaching the stopping criteria. Since the original logs are small, the stopping criteria for many of them was never reached, and the logs were fully read, yielding similarity of 1. Therefore, we repeated the experiments for the duplicated logs. The figure reports the average similarity levels, with $\alpha = 0.05$, $\delta = 0.05$, and $k = 2, 3$. As can be seen, the similarity levels obtained for all logs exceed the 95% threshold, with a moderate reduction as the log size increases. Furthermore, 97.7% of all experiments obtained an average similarity of 90% and higher, which demonstrases high reliability.

For lack of space, we do not provide the reliability levels here. The trend for the average reliability levels observed earlier was repeated in these experiments, with $\alpha = 0.01$ being too conservative and $\alpha = 0.85$ not being strict enough.

Second, we run the BEAR experiment over Log Set III. As many of the 343 transitions of the DTMC have very low frequencies, in this experiment we included low insensitivity δ values. We used $\alpha = 0.05$ and $\delta = 0.005, 0.01, 0.02, 0.05$ and repeated the experiment 10 times with each δ.

Fig. 3 shows 1-similarity values as function of a target insensitivity level δ for the transitions observed during the experiments and achieving (α, δ)-confidence. These values capture the absolute distances between the true transitions' frequencies and the estimated frequencies. As expected, these distances reduce with δ. Further, for a large majority of the transitions, the obtained distance lays below δ. The average similarity values measured are 99.72%, 99.61%, 99.49%, 98.97% for $\delta = 0.005, 0.01, 0.02, 0.05$ resp., showing a consistent increase in similarity as insensitivity decreases.

Finally, the δ-reliability levels measured (not shown in the figure) are 90.89%, 92.88%, 96.83%, 99.29% for $\delta = 0.005, 0.01, 0.02, 0.05$ resp., which illustrates that reducing insensitivity level increases the error rate of the experiments. This is explained by the fact that reducing the insensitivity level requires capturing more transitions with higher precision.

We conclude that the results demonstrate that our method reaches high reliability levels, which is reflected by the small distances (high similarity values) reported above.

[2] We did not include the `cvs.net` log in Fig. 2, as both its first and third quartiles equal 1, which shrinks the body of its boxplot to a single point.

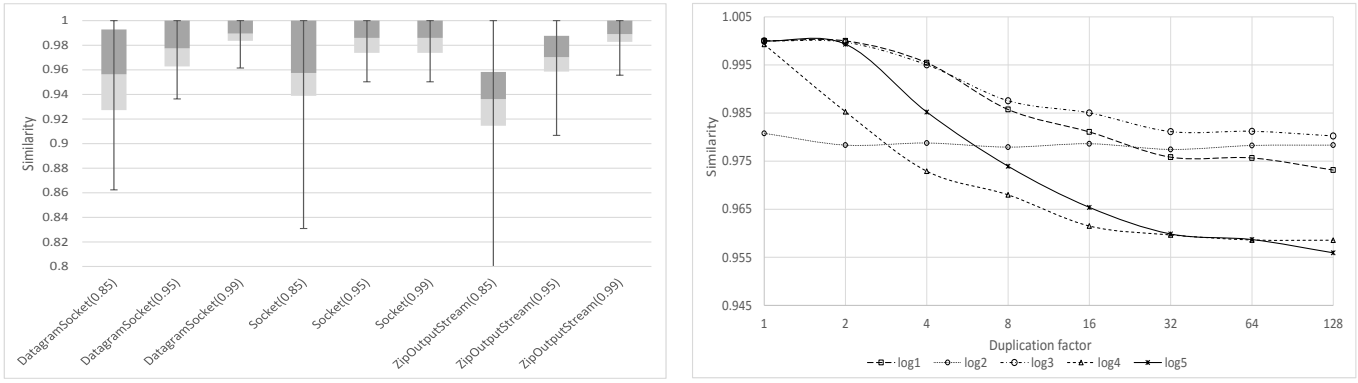

Figure 2: Similarity obtained for Log Set I and Log Set II, see Section 4.3.2.

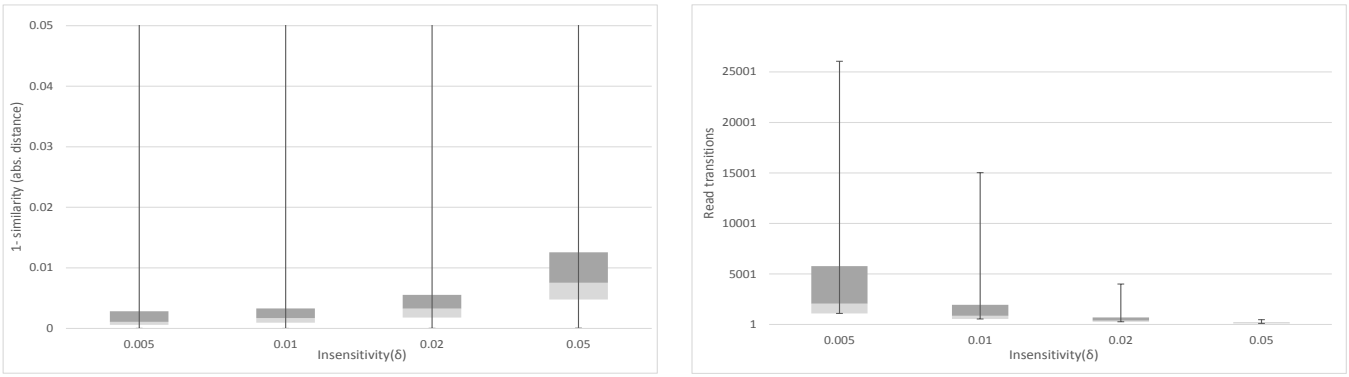

Figure 3: The figures report the results for Log Set III. (left) 1 - similarity (distance) between the true and estimated transitions' frequencies as function of δ, see Section 4.3.2. (right) Number of read transitions as function of δ, see Section 4.3.3.

Remark 8. The reader may notice that we discarded unobserved transitions from our analysis. We did so since large majority of the transitions had frequencies lower than the δ values we used. Therefore, they would nearly always be considered within a distance of δ from the estimation. This makes the similarity and reliability values so high that the influence of changes in δ can no longer be observed. By discarding these transitions, we excluded the long tail (of the transitions' distribution) from our results and facilitated their comprehension.

To answer RQ2, our method is able to obtain high similarity levels, which in the case of k-Tails, indicate that most of the information is indeed captured. Further, the expected δ-reliability level of $1-\alpha$ is obtained, when the selected confidence level is above 95%. Our experiments with BEAR provide evidence that the sampled transitions' frequencies guarantee the required similarity of δ, maintaining, as required, an error rate of α or less for $\delta = 0.02, 0.05$.

4.3.3 RQ3 and RQ4

To answer *RQ3* and *RQ4*, we run the k-Tails experiment over Log Set I and Log Set II, with the same setup and parameters as in *RQ2*. For better visualization of the data, we present the average results for the experiments with $\alpha = 0.05, \delta = 0.05$, and $k = 1, 2, 3$, and note that we obtained similar results with other values of α. We repeated all experiments 30 times.

For Log Set I, Fig. 4 (top, left) presents the percentage of read traces when reaching the desired (α, δ)-confidence. As can be seen, this increases as the complexity of the analysis increases. This is not surprising as increasing k, increases the amount of information that needs to be extracted from the log. As an example, the number of k-sequences extracted from the `DatagramSocket` log is 500, 8774, and 99259 for k=1,2,3 resp. Therefore, confidence is quickly reached for $k = 1$, but is never reached for $k = 3$, in which on average 67.5% of the traces revealed at least a single new k-sequence when they were processed. This shows that for logs with high redundancy levels (where the information can be extracted from a fairly small set of randomly selected traces), our method is able to significantly reduce the number of read traces. It also shows that when the log does not contain much redundancy, as in the case of the `DatagramSocket` and `Socket` logs, with $k = 3$, all of the traces are read. We interpret this as the cost required to ensure the high reliability levels discussed in *RQ*2. The trend was repeated with $\alpha = 0.01, 0.15$ (not shown in the figure), with a higher percentage of read traces for $\alpha = 0.01$ and a lower one for $\alpha = 0.15$, as one would expect.

Fig. 4 (top, right) presents the ratio between the sample analysis execution time and the entire log analysis execution time. Two encouraging conclusions can be derived. First, when the log contains much redundancy, our method

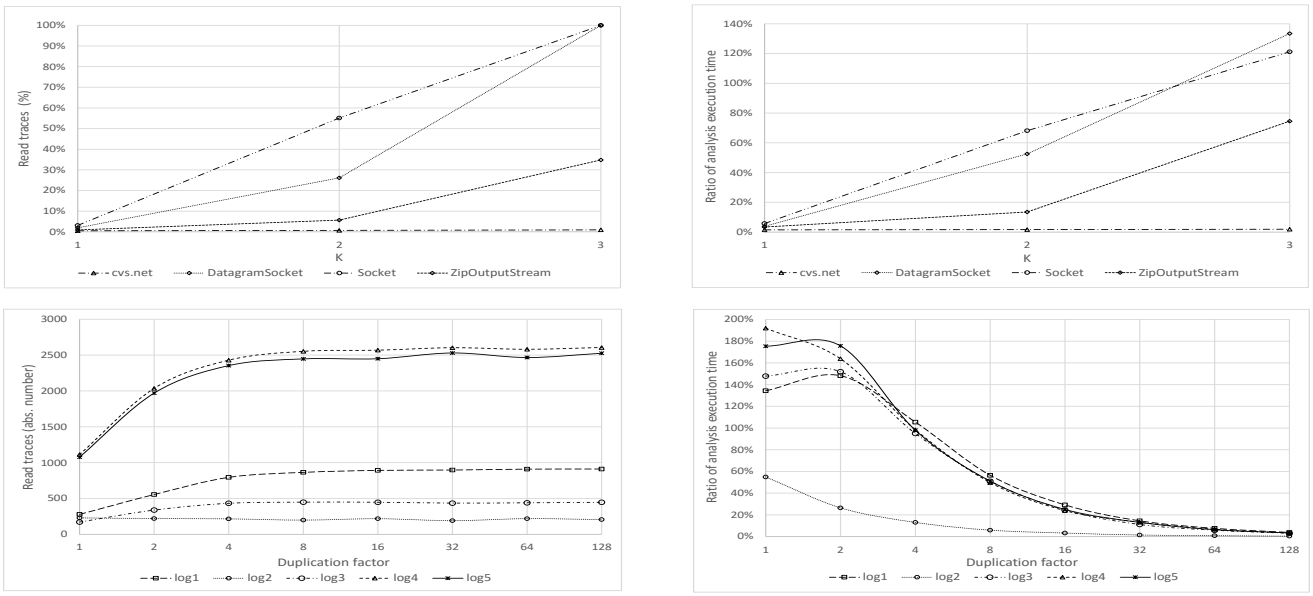

Figure 4: Read traces and execution times for Log Set I and Log Set II, see Section 4.3.3.

achieves significant reduction in execution time. As an example, when setting $k = 1$, the sample times were between 1.5% and 5.77% of the log times. Second, even in the cases where all traces are sampled, applying our method increases the required time by only a reasonable factor of 20% to 35%. Note that our implementation is rather simple and may be further optimized.

Fig. 4 (bottom, left) presents the absolute number of read traces when increasing the size of the different logs in Log Set II. Since the logs were duplicated, the same information is contained in all versions of each log. The difference between the versions of the logs is the amount of redundancy they contain. As can be seen, starting with a duplication factor of 4, the number of read traces stabilizes and does not increase with the size of the log anymore. The trend shows that our method is able to not be fooled by the redundancy and to capture the meaningful information using a constant number of traces! This justifies the claim that our method allows for sub-linear log analysis. In fact, as demonstrated by the trend in Fig. 4 (bottom, left), the analysis indeed converges to a constant sample size. Finally, Fig. 4 (bottom, right), shows the ratio between the sample times and log times for the different logs. As can be seen, the gap between these two measures widens as the log size increases. This can be explained by our earlier observation that the number of sampled traces converges to a constant (from a certain point on), in comparison with reading the entire log. One may also note that for the majority of the logs, a benefit is obtained with a duplication factor of four and more. Finally, similarly to the experiments over Log Set I, one may observe an acceptable overhead when the entire log is read, ranging from 34.4% to 92% with an average overhead of 62.4%.

Indeed, the statistical analysis does not come for free. Still, note that the entire log is read if and only if (α, δ)-confidence is not obtained. In such cases, the extra computations we perform provide the engineer with an indication that the computed model does *not* have the required statistical guarantees. We consider this to be an important, useful side contribution of our work.

To further investigate *RQ3* and *RQ4*, we run the BEAR algorithm experiment over Log Set III. To answer *RQ3*, we used the same parameters as in *RQ2*. First, we measured the number of read transitions required for each of the transitions to obtain (α, δ)-confidence. Fig. 3 (right) reports these numbers as function of δ (as before, we did not include transitions that did not obtain (α, δ)-confidence). The results show, as expected, that reducing the insensitivity level comes at the cost of reading more transitions. The average numbers of transitions read were 5796, 1684, 613, and 174 for $\delta = 0.005, 0.01, 0.02, 0.05$ resp. We also report the average number of read transitions when each run was terminated. As described in the setup, a run was terminated once all of the observed transitions achieved (α, δ)-confidence. The values were 38042, 13201, 3237, and 502 (the entire log) transitions for $\delta = 0.005$, 0.01, 0.02, 0.05 resp.

Finally, to answer *RQ4*, we present the average execution times recorded for each of the experiments. The measured times in minutes are 12.77, 4.23, 1.1 ,0.15 for $\delta = 0.005$, 0.01, 0.02, 0.05 resp. In contrast, we measured an average of 12 minutes for the original BEAR implementation, as stated in *RQ1*. This shows that our method can indeed lead to significant time reductions. Further, one can observe that even in the cases where the experiment only ended after all the transitions have been read, such as in the case of $\delta = 0.005$ (very low insensitivity), the overhead is very moderate with an average of 6.4% additional execution time. Here again, we claim that the extra computation is not for nothing, as it provides the user with important information about the reliability of the constructed model.

We conclude that the results demonstrate our method's ability to yield major reduction in the number of read transitions and in total computation time. Further, the results emphasize the trade-off between sensitivity and scalability.

To answer RQ3, after experimenting with values that were shown to achieve high reliability levels in RQ2, we conclude that our method is able to reduce the number of traces (or transitions) read. Our method effectively addresses redundancy and converges to a constant sample size, while still capturing the information available in the log. Further, the method is able to identify logs that do not contain much redundancy and in these cases read most traces (or transitions) to ensure high reliability.

To answer RQ4, our method is able to dramatically reduce the analysis time of large logs with high redundancy. It also has an acceptable overhead in cases where the entire log needs to be read, i.e., when (α, δ)-confidence is never obtained. In such cases, it indicates that more data is required to construct a more reliable model.

4.4 Threats to Validity and Limitations

We now discuss threats to validity of our answers to the research questions and additional limitations of our work.

One may argue that the duplication of logs in Log Set II may not be representative of real-world long logs. Still, as every system has a certain degree of complexity with respect to an algorithm, as logs become longer, redundancy with respect to the k-Tails algorithm is inevitable (this holds for many other behavioral log analysis algorithms as well). Our use of duplications allowed us to control for this redundancy in our experiments. That said, we have also used real-world large logs without duplications (Log Set III).

The similarity measure does not compare the final output of the selected algorithms. We have decided to focus on the data elements from which models are constructed (k-sequences, transitions) and not on the final output of the algorithms. Our decision is motivated by the fact that these elements determine the final output of the algorithms. There may be different ways on how to measure similarity between the final outputs; choosing between them may depend on the specific intended use.

The filters and criteria used to partition the log into traces, can have a dramatic effect on the entire analysis and on the resulting model (e.g., defining user classes in BEAR). For different filters and different partition criteria, one may obtain results that are different than the ones we have seen in our experiments. Note, though, that our method does not restrict the partition, but only requires that the traces which are derived from the partition (e.g., read traces in k-Tails, single events in BEAR) can be randomly and independently selected.

The analysis of traces may only take a small part of the complete analysis computation time. In this work, we focus on applying statistical means to reduce the amount of analyzed data. Clearly, there are other steps in applying a behavioral log analysis algorithm, such as parsing the log, filtering and partitioning it into traces, and constructing a model from the extracted data. These are all important practical aspects which are beyond the scope of our present paper. For example, in the present work, to obtain a random trace we read the entire log (or duplicated it in memory). As part of future work, we will explore means to randomly sample traces from a log without first reading it entirely (and without assuming a predefined log structure).

The log may not represent the behavior of the system that generated it. Generating a representative log of a system is a problem that deserves its own investigation and is beyond the scope of the present paper. In the present paper we limit our focus to investigate if behavioral log analysis can be made scalable by sampling a portion of the available log and obtaining results that are similar to the results that may be obtained by processing the entire available log, with required statistical guarantees.

The results of statistical log analysis, albeit correct, may not be useful when the given log has a long tailed distribution with many very infrequent properties. For example, in the case of the BEAR algorithm, if 90% of the transitions in the log have frequency of less than 1%, setting the insensitivity δ to be greater than 1% will cause most of the transitions to be missed completely. We do not know whether many real-world logs exhibit such long tailed distributions. Note that the distribution depends on the chosen filters and partition (see above).

5. CONTRIBUTIONS AND FUTURE WORK

In this paper we presented *statistical log analysis*, which uses 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 part in a log as a trial in an experiment. We demonstrated the application of our approach to two different analyses: the classic k-Tails algorithm and the recently introduced BEAR inference algorithm. Extensive evaluation with logs generated from real-world models and with real-world logs provided by our industrial partners provides evidence for the need for scalability and for the effectiveness of statistical log analysis.

We believe that statistical log analysis has much potential to help in scaling up existing behavioral log analysis algorithms and thus in bringing these algorithms to the hands of software engineers in practice. We consider the following future directions. First, we investigate other, more elaborated and robust stopping criteria, which can result in less conservative yet still correct analysis results. Second, together with our industrial partners, we look for additional analysis algorithms where statistical log analysis can be applied, for example, in scenario-based trigger/effect analysis [21] and in log comparisons in the context of software evolution. Lastly, we investigate a more sophisticated machinery to remove some of the underlying assumptions of our approach (i.e., time-homogeneity, incrementality).

Acknowledgements. We thank our colleagues in the unnamed multi-national telecommunications equipment company for sharing their logs with us. We thank the authors of [17] for sharing their web log data with us. We thank the anonymous reviewers of the conference for their helpful comments. Part of this work was done while SM was on sabbatical as visiting scientist at MIT CSAIL. This work has been partly supported by Len Blavatnik and the Blavatnik Family Foundation.

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] N. Busany and S. Maoz. Behavioral log analysis with statistical guarantees. In *ESEC/FSE*, pages 898–901. ACM, 2015.

[7] H. Cohen and S. Maoz. The Confidence in Our k-Tails. In *ASE*, pages 605–610, 2014.

[8] H. Cohen and S. Maoz. Have we seen enough traces? In M. B. Cohen, L. Grunske, and M. Whalen, editors, *ASE*, pages 93–103. IEEE, 2015.

[9] 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.

[10] 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.

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

[12] M. H. DeGroot and M. J. Schervish. *Probability and Statistics*. Addison Wesley, 3rd edition, 2002.

[13] M. El-Ramly, E. Stroulia, and P. G. Sorenson. From run-time behavior to usage scenarios: an interaction-pattern mining approach. In *KDD*, pages 315–324, 2002.

[14] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *TSE*, 27(2):99–123, 2001.

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

[16] M. Gabel and Z. Su. Online inference and enforcement of temporal properties. In *ICSE*, pages 15–24, 2010.

[17] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli. Mining behavior models from user-intensive web applications. In *ICSE*, pages 277–287, 2014.

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

[19] C. Lee, F. Chen, and G. Rosu. Mining parametric specifications. In *ICSE*, pages 591–600, 2011.

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

[21] D. Lo and S. Maoz. Mining scenario-based triggers and effects. In *ASE*, pages 109–118, 2008.

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

[23] 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.

[24] 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.

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

[26] C. Luo, F. He, and C. Ghezzi. Inferring Software Behavioral Models with MapReduce. In *Proc. of SETTA*, volume 9409 of *LNCS*, pages 135–149. Springer, 2015.

[27] L. Mariani, S. Papagiannakis, and M. Pezzè. Compatibility and regression testing of COTS-component-based software. In *ICSE*, pages 85–95, 2007.

[28] L. Mariani and M. Pezzè. Behavior capture and test: Automated analysis of component integration. In *ICECCS*, pages 292–301, 2005.

[29] 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.

[30] M. Pradel and T. R. Gross. Automatic generation of object usage specifications from large method traces. In *ASE*, pages 371–382, 2009.

[31] J. Quante and R. Koschke. Dynamic protocol recovery. In *WCRE*, pages 219–228, 2007.

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

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

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

[35] S. Wang, D. Lo, L. Jiang, S. Maoz, and A. Budi. Scalable Parallelization of Specification Mining. In C. Bird, T. Menzies, and T. Zimmermann, editors, *The Art and Science of Analyzing Software Data*. Morgan Kaufmann, 2015.

[36] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal API rules from imperfect traces. In *ICSE*, pages 282–291, 2006.