The Confidence in our k-Tails

Hila Cohen and Shahar Maoz

School of Computer Science
Tel Aviv University, Israel

ABSTRACT

k-Tails is a popular algorithm for extracting a candidate behavioral model from a log of execution traces. The usefulness of k-Tails depends on the quality of its input log, which may include too few traces to build a representative model, or too many traces, whose analysis is a waste of resources. Given a set of traces, how can one be confident that it includes enough, but not too many, traces? While many have used the k-Tails algorithm, no previous work has yet investigated this question.

In this paper we address this question by proposing a novel notion of log completeness. Roughly, a log of traces, extracted from a given system, is k-complete, iff adding any new trace to the log will not change the resulting model k-Tails would build for it. Since the system and its full set of traces is unknown, we cannot know whether a given log is k-complete. However, we can estimate its k-completeness. We call this estimation *k-confidence*.

We formalize the notion of k-confidence and implement its computation. Preliminary experiments show that k-confidence can be efficiently computed and is a highly reliable estimator for k-completeness.

General Terms

Algorithms, Design, Experimentation

Keywords

Dynamic specification mining; probabilistic approach

1. INTRODUCTION

Dynamic specification mining algorithms extract candidate specifications from logs of execution traces. One such well-known algorithm is k-Tails, first introduced in [6]. k-Tails input consists of a set of traces and a positive number k. Its output is a finite state automaton, approximating the behavior of the system from which the traces have been extracted. It works by starting with a most refined model and iteratively coarsening it by merging states whose future k states are equivalent. Over the last two decades, the algorithm has been implemented and used, in many variants, e.g., [4, 7, 17, 21, 22, 28].

The usefulness of a specification mining algorithm in general and of k-Tails in particular depends on the quality of its input log. On the one hand, if there are too few traces, the constructed model may not be a good representation of the behavior of the system under investigation. On the other hand, if there are many traces, perhaps some are redundant, and executing the algorithm over all of them is a waste of resources. Given a set of traces, how can one know whether it includes enough, but not too many, traces, to build a representative model? While many have used the k-Tails algorithm, no previous work has yet investigated this question.

In this paper we address this question by proposing a novel notion of log completeness. Roughly, a log of traces, extracted from a given system, is k-complete, iff adding any new trace to the log will not change the resulting model k-Tails would build for the log. Since the system and its full set of traces is unknown, we cannot know whether a given log is k-complete. However, we can estimate the probability that the log is k-complete. We call this estimation *k-confidence*.

A low k-confidence of, say, 0.3, hints that the model which may be built by the k-Tails algorithm is probably far from characterizing the behavior of the system under investigation. There is high probability that additional traces will change the model which k-Tails would build. Given a log with such a low k-confidence, one should better look for additional traces. A very high k-confidence, of say, 0.95, hints that the model which may be built by the k-Tails algorithm is probably very close to correctly characterize the behavior of the system under investigation. There is very low probability that additional traces will change this model.

We compute the k-confidence of a set of traces, by estimating the probability that it includes all possible sequences of length k that the system under investigation can generate. The estimation is based on the appearances of the sequences in the specific traces we see in the log and on their frequencies. It does not require running the k-Tails algorithm.

An interesting and important feature of k-confidence is its non-monotonicity. Specifically, as traces are added to a log, its k-confidence may increase or decrease. Indeed, in the absence of additional information, e.g., about the order in which traces are produced, a new trace may introduce new information, which should lead us to revise our previous estimations. Note, however, that the k-confidence of a log does not depend on the order of traces in the log.

ASE'14, September 15-19, 2014, Vasteras, Sweden.

http://dx.doi.org/10.1145/2642937.2642944.

We have implemented the computation of k-confidence and evaluated it over several models. The evaluation shows that k-confidence can be efficiently computed and is a highly reliable estimator for k-completeness. Thus, it can be effectively used by engineers using the k-Tails algorithm.

The next section presents two examples. Sect. 3 presents the formal definitions and the computation of k-confidence. Sect. 4 presents a preliminary evaluation and Sect. 5 discusses design decisions and limitations of our work. Sect. 6 discusses related work and Sect. 7 concludes.

2. EXAMPLES

We use an example to demonstrate k-confidence usage and another one as a running example for this paper. The traces and models described below are available in [1].

We consider the example CVS client, presented by Lo and Khoo in [16]. The client provides the following interaction scenarios: initialization, multiple-file upload, download, and deletion, multiple-directory creation and deletion. Consider an engineer having a log of 23 traces from this system, trying to extract a model of its behavior. Fig. 1 shows the model suggested by k-Tails with $k = 1$ for this log (the dashed transitions `login` to `storeFile`, `rename` to `logout` and `removeDir` to `removeDir` are *not* part of the model suggested by k-Tails for this 23 traces log). Can she be confident that this model is a good representation of the system's behavior? Perhaps the model is partial and more traces are needed? Indeed, adding 25 more traces, and feeding the resulting log of 48 traces to k-Tails, results in a revised model as shown in Fig. 1 (including the dashed transitions).

Our tool computes a k-confidence of 0.36 to the first log of 23 traces and a k-confidence of 0.96 to the second log of 48 traces (for $k = 2$). These k-confidences are essentially the probabilities that the logs are k-complete, computed solely based on the traces themselves.

On the one hand, if the engineer has set a minimum k-confidence threshold of 0.95 (which we suggest to be a reasonable choice), she would not have stopped analyzing traces too soon, i.e., before finding the additional transitions.

On the other hand, should the engineer continue to analyze more traces of this system? Given the computed k-confidence of 0.96, the probability that additional traces will reveal new behaviors is very small. Indeed in our example, an extension of the log with 28 additional traces (many of them new traces, not duplicates of any of the traces seen before), resulted in a slightly higher k-confidence of 0.99 but in the *same* model as was suggested by k-Tails for the second log of 48 traces. Thus, by stopping the analysis when k-confidence passed the 0.95 threshold, the engineer saved the resources required in order to produce and analyze the additional traces, yet did not lose any information.

As a running example for this paper we consider a simpler model representing a file reading protocol. The model and 8 randomly generated traces from it are shown in Fig. 2.

3. K-COMPLETENESS AND K-CONFIDENCE

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 $e_1, \ldots, e_m \in \Sigma$. For $j \geq 1$ we use $\sigma(j)$ to denote the jth element in σ.

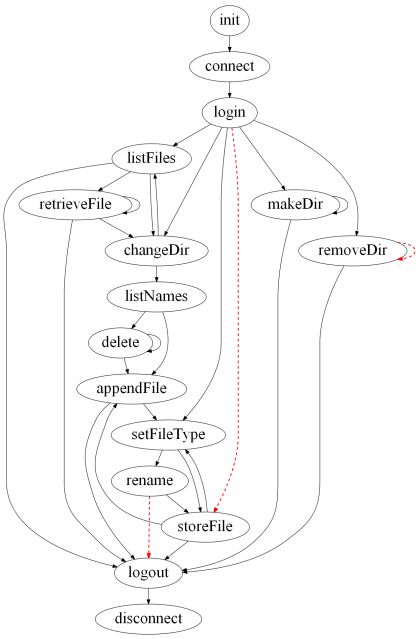


Figure 1: CVS model as mined by k-Tails from the log of 23 traces (without the dashed transitions) and from the log of 48 traces (including the dashed transitions)

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 $L(M)$.

We characterize the k-Tails algorithm using a property we call 'k-directly-follows'. Roughly, the property 'k-directly-follows' for a sequence of events $es = \langle e_1, e_2, \ldots, e_k \rangle$ holds in a trace iff the sequence appears somewhere in the trace.

We formalize the 'k-directly-follows' property using a pair of functions. The first function K_{tr} maps every trace σ and a sequence es to a value in $\{0, 1\}$ (intuitively, K_{tr} assigns a value to the relation between the k-directly-follows property and the trace). The second function K_{log} maps subsets of $\{0, 1\}$ to $\{0, 1\}$, aggregating the results of K_{tr} per sequence, from the trace level to the log level.

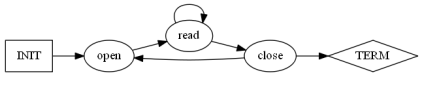
The formal definitions of the two functions follow the semantics of 'k-directly-follows'. Formally:

Definition 1 (k-directly-follows property)**.**

$$K_{tr}(\sigma, \langle e_1, e_2, \ldots, e_k \rangle) = \begin{cases} 1 & \exists_j \bigwedge_{1 \leq m \leq k} \sigma(j + m - 1) = e_m \\ 0 & otherwise \end{cases}$$

$$K_{log}(S) = \begin{cases} 1 & 1 \in S \\ 0 & otherwise \end{cases}$$

Example 1. For tr_1 and tr_3 shown in Fig. 2, $k = 2$, and the sequences of length k $\langle open, read \rangle$, $\langle read, read \rangle$, $\langle open, open \rangle$ we have
$K_{tr}(tr_1, \langle open, read \rangle) = 1$, $K_{tr}(tr_3, \langle open, read \rangle) = 1$,
$K_{tr}(tr_1, \langle read, read \rangle) = 0$, $K_{tr}(tr_3, \langle read, read \rangle) = 1$,
$K_{tr}(tr_1, \langle open, open \rangle) = 0$, $K_{tr}(tr_3, \langle open, open \rangle) = 0$.
For a log $\{tr_1, tr_3\}$ and the same three sequences we have
$K_{log}(\{K_{tr}(tr_1, \langle open, read \rangle), K_{tr}(tr_3, \langle open, read \rangle)\}) = 1$
$K_{log}(\{K_{tr}(tr_1, \langle read, read \rangle), K_{tr}(tr_3, \langle read, read \rangle)\}) = 1$
$K_{log}(\{K_{tr}(tr_1, \langle open, open \rangle), K_{tr}(tr_3, \langle open, open \rangle)\}) = 0$.



tr_1:open,read,close
tr_2:open,read,read,close
tr_3:open,read,read,close,open,read,read,close,open,read,close
tr_4:open,read,read,close
tr_5:open,read,read,close,open,read,read,read,read,read,close,open,read,close
tr_6:open,read,close,open,read,close
tr_7:open,read,close,open,read,read,close,open,read,close
tr_8:open,read,read,close

Figure 2: A simple example model and 8 randomly generated traces

3.2 K-Completeness

We say that a log $l \in L(M)$ is k-complete if the information one may extract about the k-directly-follows property from the log l is equal to the information one may extract about this property from any log that includes l (and thus specifically from all the traces in $T(M)$). Formally:

Definition 2 (k-completeness)**.** A log $l \in L(M)$ is k-complete iff $\forall l' \in L(M)$ s.t. $l \subseteq l'$, $\forall es \in \Sigma^k$
$K_{log}(\{K_{tr}(\sigma, es)|\sigma \in l\}) = K_{log}(\{K_{tr}(\sigma, es)|\sigma \in l'\})$.

The intuition behind this definition is as follows. Consider a model produced by k-Tails for a log, and a new trace whose all subtraces of length $k+1$ appear in the produced model. Applying k-Tails to a new log that consists of the original log and the new trace, produces the same model.

Example 2. For the logs $l = \{tr_1\}$, $l' = \{tr_1, tr_3\}$, and the sequence $\langle read, read\rangle$: $K_{log}(\{K_{tr}(\sigma, \langle read, read\rangle)|\sigma \in l\}) \neq K_{log}(\{K_{tr}(\sigma, \langle read, read\rangle)|\sigma \in l'\})$. Thus, $l = \{tr_1\}$ is not 2-complete.

3.3 K-Confidence: Estimating K-Completeness

For a fixed number k and a log $l \in L(M)$, our goal is to compute the probability that l is k-complete, i.e., to compute l's k-confidence. We define a random variable $Y(\sigma)$ over $\Omega = T(M)$, which maps a trace to its k-directly-follows property results. Formally:

$$Y(\sigma) : Y(\sigma)[es] = K_{tr}(\sigma, es).$$

Example 3. To continue our example from Fig. 2, the trace tr_1 and the sequences $\langle open, read\rangle$ and $\langle read, read\rangle$, we have $Y(tr_1)[\langle open, read\rangle] = 1$ and $Y(tr_1)[\langle read, read\rangle] = 0$.

For $y \in \{0,1\}^{|\Sigma|^k}$ we denote the probability that Y equals y by $\pi(y)$:

$$\pi(y) = \mathbb{P}[Y = y].$$

y can be viewed as a k-dimensional array over $\{0,1\}$.

$\pi(y)$ is the probability that in a random trace from $T(M)$ we get the values of the k-directly-follows property as they are encoded in y. It is determined by M but M is considered unknown.

We consider all traces from a log l to be samples from Y. We assume that traces are randomly and independently chosen from $T(M)$. If $|l| = n$ we denote them by $Y_1, Y_2, \ldots, Y_n$. These are independent, identically distributed random variables, versions of Y. Another random variable we define is Y^n, which aggregates all these samples to the k-directly-follows values of the entire log:

$$Y^n : Y^n[es] = K_{log}(\{Y_i[es]|1 \leq i \leq n\})$$

We now define the true, but unknown, k-directly-follows values, $f(\pi)$, in order to later compute the probability that Y^n is equal to it:

$$f(\pi) : f(\pi)[es] = K_{log}(\{y[es]|\pi(y) > 0\}).$$

Example 4. For the model shown in Fig. 2 and the sequence $\langle read, read\rangle$, since there are traces in $T(M)$ where the sequence appears and others where it does not appear, we have $f(\pi)[\langle read, read\rangle] = K_{log}(\{y[\langle read, read\rangle]|\pi(y) > 0\}) = K_{log}(\{0,1\}) = 1$. For the same model and the seqeunce $\langle read, open\rangle$, since the sequence does not appear in any trace in $T(M)$, we have $f(\pi)[\langle read, open\rangle] = K_{log}(\{y[\langle read, open\rangle]|\pi(y) > 0\}) = K_{log}(\{0\}) = 0$.

Note that M determines $f(\pi)$ (it is independent of any specific log). We can now write Def. 2 using the above notation as follows: a log l of size n is k-complete iff

$$Y^n = f(\pi).$$

Recall that our goal is to compute the probability that l is k-complete. Using the notation defined above, what we are looking for is $\mathbb{P}[Y^n = f(\pi)]$.

Definition 3 (k-confidence)**.** The k-confidence of a log of size n, $l \in L(M)$ is $\mathbb{P}[Y^n = f(\pi)]$.

3.4 Computing K-Confidence

We use es to denote the sequence $\langle e_1, e_2, \ldots, e_k\rangle$. Since it is fixed, we omit π from the formulas below.

The probability that the k-directly-follows property does not hold in the model but appears in one of the traces is zero. Thus, we only need to consider the other case, where the sequence of length k does not appear in the traces ($Y^n[es] = 0$) but is possible in the model ($f[es] = 1$). Formally:

$$\begin{aligned}\mathbb{P}[Y^n = f] &= 1 - \mathbb{P}[\exists es. Y^n[es] = 0 \wedge f[es] = 1] \\ &\geq 1 - \sum_{es} \mathbb{P}[Y^n[es] = 0 \wedge f[es] = 1]\end{aligned}$$

$$\begin{aligned}\mathbb{P}[Y^n[es] = 0 \wedge f[es] = 1] &= \begin{cases} \mathbb{P}[Y^n[es] = 0] & f[es] = 1 \\ 0 & f[es] = 0 \end{cases} \\ &= \begin{cases} \prod_{1 \leq i \leq n} \mathbb{P}[Y_i[es] = 0] & f[es] = 1 \\ 0 & f[es] = 0 \end{cases}\end{aligned}$$

We use q_{es} to denote the probability that the k-directly-follows property for es holds on a random trace from $T(M)$, i.e., that the sequence $\langle e_1, e_2, \ldots, e_k\rangle$ appears somewhere in the trace. When $f[es] = 1$ we have $q_{es} > 0$ and $\prod_{1 \leq i \leq n} \mathbb{P}[Y_i[es] = 0] = (1 - q_{es})^n$.

Since q_{es} is unknown, we estimate it using the average of the n random variables Y_i

$$\hat{q}_{es} = \sum_{i=1}^{n} \frac{Y_i[es]}{n} \tag{1}$$

and so overall, we have

$$\mathbb{P}[Y^n = f] \geq 1 - \sum_{\{es|\hat{q}_{es} > 0\}} (1 - \hat{q}_{es})^n. \tag{2}$$

es	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$
`close;read`	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)
`close;open`	0.00 (-0.00)	0.00 (-0.00)	0.33 (-0.30)	0.25 (-0.32)	0.40 (-0.08)	0.50 (-0.02)	0.57 (-0.00)	0.50 (-0.00)
`open;read`	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)
`open;open`	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)
`open;close`	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)
`read;open`	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)
`read;read`	0.00 (-0.00)	0.50 (-0.25)	0.67 (-0.04)	0.75 (-0.00)	0.80 (-0.01)	0.67 (-0.00)	0.71 (-0.00)	0.75 (-0.00)
`read;close`	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)	1.00 (-0.00)
`close;close`	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)	0.00 (-0.00)
2-confidence	1	0.75	0.67	0.68	0.92	0.98	0.99	0.99

Table 1: Example k-confidence computation for the log of 8 traces shown in Fig. 2, with $k = 2$. If we are using a 0.95 threshold, we stop the computation after 6 traces have been analyzed. See Sect. 3.5.

3.5 Example Computation

We demonstrate a k-confidence computation on our running example model and generated traces shown in Fig. 2.

For these traces, Table 1 shows the computation of k-confidence for $k = 2$. Each row corresponds to a different sequence *es*, e.g., the first row corresponds to the property '`close` directly follows `read`'. Each column corresponds to the addition of a new trace to the accumulating log. The table cell for row *es* and column $n = j$ shows the value $\hat{q}_{es}$ from Equ. 1. At the bottom line of each column we show the accumulating total $\mathbb{P}[Y^n = f]$, which represents the probability that the set of traces analyzed so far is k-complete, i.e., k-confidence as defined in Equ. 2. Thus, the table shows the intermediate k-confidence values.

As an example, for $n = 3$, the computed value $\hat{q}_{\texttt{close;open}} = 0.33$ (see Equ. 1) is the probability to have an instance of `close;open` in a random trace, given the 3 traces analyzed so far. Since this is the third trace already, the negative contribution to the accumulating k-confidence is $(1-0.33)^3 = 0.30$ (see Equ. 2). That's the probability that in a random log of size 3, `close;open` will not occur.

Given a threshold of 0.95, in this example one could stop after analyzing only 6 traces from the original log, and have high confidence that the model built by k-Tails for this 6 traces log will not change if additional traces are considered.

Note that in our example, as traces are added, k-confidence may increase or decrease. In fact, the first computed k-confidence is (always) already 1; however, in general, the larger the log, the lesser the expected decrease in k-confidence when a new trace is added (we demonstrate it in this example by showing the computed values for $n = 7$ and $n = 8$). In Sect. 5 we discuss this non-monotonicity and the need for a minimal number of traces to produce useful results.

4. EVALUATION

We have implemented the computation of k-confidence. The implementation gets as input a log (a set of traces) and a number k. Note that the k-confidence computation does not need to run the k-Tails algorithm. Given a log l, for each trace $\sigma \in l$, and for each sequence of length k *es*, we compute $K_{tr}(\sigma, es)$. The computed values are used as input for the computation of the log's k-confidence.

We conducted preliminary experiments to check whether k-confidence can be used as an effective proxy for true k-completeness. All logs, models, and implementation code described in this paper are available for inspection and reproduction together with documentation from [1].

4.1 Methodology

We use two measures to evaluate the effectiveness of k-confidence, k-reliability and k-redundancy. We define the k-reliability of a log to be 1 if the log is k-complete and 0 otherwise. For a set of logs L, a mean k-reliability close to 1 hints that most of the logs are k-complete. We define the k-redundancy of a log to measure how close is it to its minimal prefix log which is k-complete (assuming an arbitrary fixed order of traces). For a set of logs L, a mean k-redundancy close to 0 and a low standard deviation hint that the logs do not include much redundant traces. Formally:

Definition 4 (k-reliability)**.** The k-reliability of a log l is $rel(l) = \begin{cases} 1 & l \text{ is k-complete} \\ 0 & otherwise \end{cases}$.

Definition 5 (k-redundancy)**.** Given a log l (in a fixed arbitrary order), let $i_{min}(l)$ be the minimal index of traces in l such that the set of traces $\sigma_1, \sigma_2, \ldots, \sigma_{i_{min}} \in l$ is k-complete. The k-redundancy of l is $red(l) = 1 - \frac{i_{min}(l)}{|l|}$.

Example 5. Consider our running example and the results of computing its 2-confidence (Fig. 2 and Table 1). This log's 2-reliability is $rel(l) = 1$. Since it reaches 2-completeness already after the 3rd trace and we stopped the computation when reaching the 0.95 threshold after the 6th trace, its 2-redundancy is $red(l) = 1 - 3/6 = 0.5$.

Note that to calculate k-reliability and k-redundancy, as in the above example, one must know the system from which the traces were extracted, so that she can compute the true value for each sequence. This is typically unknown in a real-world setting, but it is known in our controlled evaluation.

4.2 Experiment Design

For each model we used the following experiment protocol. We first generated traces from the model using the trace generator of [21] with high state coverage. We then created an initial log by randomly selecting a minimal number of 10 traces, and iteratively computed the current log's k-confidence and added a trace to it. We kept adding traces to the current log until we reached a fixed k-confidence threshold of 0.95 (or we ran out of traces to add). Finally, we computed the k-reliability and k-redundancy of the final log (we checked true k-completeness using a model-checker, i.e., by expressing the k-directly-follows properties in temporal logic and verifying them against the model). We used $k = 2$.

We repeated the above protocol for each model 200 times and computed the mean of k-reliability and k-redundancy for the sets of 200 logs.

Model	$\|\Sigma\|$	# s	# t	a.l.
java.net.DatagramSocket	29	9	82	15
java.net.MultiCastSocket	16	7	36	12
java.net.Socket	42	16	209	21
java.net.URL	17	7	61	9
java.util.Formatter	8	6	15	7
java.util.StringTokenizer	7	6	18	2

Table 2: Models used in our evaluation, taken from Pradel et al. [25]. For each model we report the size of the alphabet Σ, the number of states (# s), the number of transitions (# t), and the average length of the generated traces used in the logs (truncated).

Model	$\|l\|$	c. m	rel. m	red. m/sd
java.net.DatagramSocket	1121	0.94	1	0.53/0.10
java.net.MultiCastSocket	467	0.97	0.99	0.63/0.17
java.net.Socket	1891	0.83	0.98	0.39/0.16
java.net.URL	509	0.96	1	0.58/0.11
java.util.Formatter	120	0.96	0.99	0.62/0.16
java.util.StringTokenizer	1808	0.97	0.94	0.65/0.22

Table 3: Experiment results. Each experiment was repeated 200 times for each of the models. For each model we report the average number (truncated) of traces required to reach k-confidence > 0.95, the mean of the calculated k-confidence (c. m), k-reliability (rel. m), and the mean and standard deviation of k-redundancy (red. m/sd). Traces generated using the trace generator of [21].

4.3 Results

We used 6 finite-state automaton models from Pradel et al. [25]. Table 2 lists the models we used. For each model we report the size of the alphabet Σ, the number of states and transitions, and the average length of the generated traces used in the logs. Table 3 shows the experiment results on the models listed in Table 2.

The results show that **our k-confidence computation is highly reliable**. In only one case (java.net.Socket) we ran out of traces to add before k-confidence reached the 0.95 threshold. **On the other hand, k-redundancy is not always low**, which hints to the conservative nature of the k-confidence definition.

All experiments were executed on an ordinary laptop computer, Intel i7 CPU 3.0GHz, 8GB RAM with Windows 7 64-bit OS, Java 1.7.0_09 64-bit. For all models, in all our experiments, k-confidence computation never exceeded 15 milliseconds. **This shows that the k-confidence computation is fast**. It is not surprising as the computation is, by definition, linear in the number of traces in the log.

4.4 Threats to Validity

We now discuss threats to the validity of our results. First, the selection of models in our evaluation may not represent typical systems. We used publicly available models taken from [25]. Yet, we do not know if these are representative of real-world systems.

Second, in our evaluation we used a publicly available trace generator, from [21], with high state coverage. It is possible that one may get different results if a different trace generator or a different coverage criteria are used.

Finally, the selection and order of traces in the log affect the point where the analysis may reach the confidence threshold and thus affect the point used to compute redundancy (although, by definition, the k-confidence and k-completeness of a log do *not* depend on the order of traces in it). We mitigated this by using randomization in the selection of traces and in the order in which they were analyzed, and by repeating all evaluation experiments 200 times.

5. DISCUSSION AND LIMITATIONS

We now discuss some important design decisions and several limitations of our present work.

An important assumption underlying our work is that the traces are randomly and independently chosen from $T(M)$. If this is not the case, e.g., if one uses a biased trace generator which favors some system features over others, and the bias is *known*, it may be possible to account for this bias in the computation of the q_{es}s and improve it.

One may be concerned about the non-monotonicity of our notion of k-confidence; as traces are added to a log, its k-confidence may increase or decrease. However, we consider this non-monotonicity an advantage; in the absence of additional information, e.g., about the order in which traces are produced, a new trace may introduce new information (i.e., reveal a new k-length sequence), which should lead us to revise our previous estimations (see, e.g., in Table 1, the confidence decreases when the 3rd trace introduces the sequence `close;open`). Moreover, as n grows, the probability that a sequence that is possible in the system has not yet been seen decreases, and so, roughly, the larger the log, the lesser the expected decrease in k-confidence when a new trace is added, and the expectation of a decrease approaches zero at the limit (a formalization and proof for this claim is outside the scope of this paper). Still, in contrast, it is important to note that our notion of k-completeness is monotonic, i.e., by definition, any extension of a k-complete log is k-complete. Also, by its definition, the k-confidence of a log does not depend on the order of traces in it.

Finally, another limitation relates to the size of the log. Our experience shows that for very small logs, e.g., under 5 traces, k-confidence results are very sensitive and fluctuate much, so they are practically useless. In general, the larger the alphabet, the more traces are required in order to get useful results (and reduce the risk that additional traces decrease the computed k-confidence). Still, note that this limitation is typically not a problem in practice because real-world logs consist usually of many traces.

6. RELATED WORK

Dynamic approaches to specification mining look for candidate specifications in execution traces (see, e.g., [5, 9–15, 19, 20, 22, 23, 26, 27, 29, 30]). None of these, to the best of our knowledge, has considered the confidence one could have in the completeness of the input logs.

In [8], Dallmeier et al. consider dynamic specification mining and ask the following question: what makes us believe that we have seen sufficiently many executions? Indeed, it seems that in our present work we ask a similar question. However, our work and Dallmeier et al. work differ fundamentally. Dallmeier et al. look for the answer in test case generation and in static specification mining. In their work, a partial set of traces is enriched in order to explore previously unobserved aspects of the execution space, including more general behavior and more exceptional behavior. In contrast, we consider a black box setting and address this question by providing a formal probabilistic measure to the notion of 'sufficiently many traces'. We do not suggest ways to make an incomplete log more complete. Instead, we provide a probabilistic measure of k-confidence, which allows engineers who use the k-Tails algorithm to assess the k-completeness of the traces they have.

Hee et al. [24] presented a probabilistic approach to log completeness in the context of the α-algorithm [3], which mines Petri nets, in particular workflow nets [2], and is used extensively in the field of process mining. Our work is inspired by the work of Hee et al. and extends it: we present a notion of log confidence for the well known k-Tails algorithm, with preliminary evaluation of its effectiveness in the context of real-world models.

To the best of our knowledge, no other work has considered the question of estimating log completeness with regard to k-Tails specifically or with regard to other dynamic specification mining algorithms.

7. CONCLUSION AND FUTURE WORK

We presented k-confidence, as a novel means to estimate the k-completeness of a log of traces, i.e., whether additional traces may change the model that k-Tails builds for the log.

Our preliminary implementation and evaluation show that k-confidence is an effective proxy for true k-completeness. Thus, it can be used to help engineers decide whether additional traces are required. No other work in the area of specification mining has addressed this question before.

We are working on a major extension and generalization of the present work, which goes beyond the k-Tails algorithm and defines a black-box probabilistic framework with a general notion of log confidence in the context of dynamic specification mining. We plan to apply this framework to additional dynamic specification mining algorithms, e.g., Synoptic [5] and mining of scenario-based triggers and effects [18]. We hope to report on this framework in a future paper.

8. ACKNOWLEDGMENTS

Hila Cohen was supported in part by the Israel Science Foundation (grant No. 476/11).

9. REFERENCES

[1] Supporting materials on log completeness. http://smlab.cs.tau.ac.il/logcompleteness.

[2] Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1), 1998.

[3] Wil M. P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9), 2004.

[4] 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*, 2007.

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

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

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

[8] 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), 2012.

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

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

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

[12] D. Fahland, D. Lo, and S. Maoz. Mining branching-time scenarios. In *ASE*, 2013.

[13] M. Gabel and Z. Su. Online inference and enforcement of temporal properties. In *ICSE*, 2010.

[14] S. Kumar, S.-C. Khoo, A. Roychoudhury, and D. Lo. Mining message sequence graphs. In *ICSE*, 2011.

[15] C. Lee, F. Chen, and G. Rosu. Mining parametric specifications. In *ICSE*, 2011.

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

[17] D. Lo and S.-C. Khoo. SMArTIC: towards building an accurate, robust and scalable specification miner. In *SIGSOFT FSE*, 2006.

[18] D. Lo and S. Maoz. Mining scenario-based triggers and effects. In *ASE*, 2008.

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

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

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

[22] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *ICSE*, 2008.

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

[24] Kees M. van Hee, Z. Liu, and N. Sidorova. Is my event log complete? - a probabilistic approach to process mining. In *RCIS*, 2011.

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

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

[27] J. Quante and R. Koschke. Dynamic protocol recovery. In *WCRE*, 2007.

[28] S. P. Reiss and M. Renieris. Encoding program executions. In *ICSE*, 2001.

[29] N. Walkinshaw and K. Bogdanov. Inferring finite-state models with temporal constraints. In *ASE*, 2008.

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