Using entropy measures for comparison of software traces

A. V. Miranskyy,1,2,∗ M. Davison,1,3 † M. Reesor,1 ‡ and S. S. Murtaza4,§

1 Department of Applied Mathematics, University of Western Ontario, London, Ontario N6A 5B7, Canada
2 IBM Toronto Software Lab, Markham, Ontario L6G 1C7, Canada
3 Department of Statistical & Actuarial Sciences, University of Western Ontario, London, Ontario N6A 5B7, Canada
4 Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada

The analysis of execution paths (also known as software traces) collected from a given software product can help in a number of areas including software testing, software maintenance and program comprehension.

In this paper we study the applicability of the Shannon entropy and three extended entropies (Landsberg-Vedral, Rényi, and Tsallis) to the classification of traces related to various software defects. Our validation study shows the three extended entropies, with parameters chosen to emphasize rare events, show good performance.

I. INTRODUCTION

A software execution trace can be thought of as a log of information captured during any particular execution run of software. For example, a trace in Figure 1 shows the program flow entering function f1; calling f2 from f1; f2 recursively calling itself, and, eventually, exiting these functions. In order to capture this information, each function in the software is instrumented to log entry and exit points to a function.

1 f1 entry
  2 | f2 entry
  3 | | f2 entry
  4 | | f2 exit
  5 | f2 exit
  6 f1 exit

FIG. 1. An example of a trace

The comparison of program execution traces is important for a number of problem areas in software development and use. In the area of testing, for example, such comparisons can be used to: 1) determine how well user execution paths (traces collected in the field) are covered in testing [1–3]; 2) detect anomalous behavior arising during a component’s upgrade or reuse [4]; 3) map and classify defects [5–7]; 4) determine redundant test cases executed by one or more test teams [8]; and 5) prioritize test cases (to maximize execution path coverage with a minimum number of test cases) [9, 10]. Trace comparisons are also used in operational profiling (for instance, in mapping the frequency of execution paths used by different user classes) [8] and intrusion analysis (e.g., detecting deviations of field execution paths from expectations) [11].

For some problems, such as test case prioritization, traces gathered in a condensed form (such as a vector of executed function names or caller-callee pairs) are adequate [9]. However, for others, such as the detection of missing coverage and anomalous behavior using state machines, detailed execution paths are necessary [1, 4]. The time required for analyzing traces can sometimes be extremely important. For instance 1) a customer support analyst using traces to map a reported defect onto an existing set of defects to identify the problem’s root cause and advise a customer on how to fix her problem, and 2) a development analyst working with the testing team to identify missing coverage that resulted in a field defect.

Many trace comparison techniques are not scalable [12]. Based on our experience, support personnel of a large-scale industrial application with hundreds of thousands of installations can collect tens of thousands of traces per year. Moreover, a trace collected on a production system is populated at a rate of millions of records per minute.

The described need to compare traces, together with a lack of reliable and scalable tools for doing this, motivated us to investigate alternate solutions. To speed up trace comparisons, we propose that traces first be filtered out from the given set, rejecting those that are not going to match with the test cases, allowing just the remaining few to be compared for target purposes. The underlying assumption (based on our practical experience) is that most traces are not even close to being similar, just a few are even similar, and only a very few are identical.

This strategy is implemented and validated in the Scalable Iterative-unFolding Technique (SIFT) [12]. The collected traces are first compressed into several levels prior to comparing them. Each level of compression uses a unique signature, which we call a “fingerprint.”

∗ amiransk@uwo.ca
† mdavison@uwo.ca
‡ mreesor@uwo.ca
§ smurtaza@csd.uwo.ca

1 The fingerprint of the next iteration always contains more information than the fingerprint of the previous iteration, hence the term unfolding.
Starting with the highest compression level, the traces are compared, and unmatched ones are rejected. Iterating through the lower levels until the comparison process is complete leaves only traces that match at the lowest (or uncompressed) level. The SIFT objective ends here. The matched traces can then be passed on to external tools for further analysis such as defect or security breach identification.

The process of creating a fingerprint can be interpreted as a map from the very high dimensional space of traces to a low, ideally one-dimensional, space. Simple examples of such fingerprints are 1) the total number of unique function names in a trace and 2) the number of elements in a trace. However, while these fingerprints may be useful for our purposes, neither are sufficient. The “number of unique function names” fingerprint doesn’t discriminate enough: many quite dissimilar traces can share the same function names called. At the other extreme, the number of elements in the trace discriminates too much – traces which are essentially similar may have varying numbers of elements. The mapping should be such that projections of traces of different types should be positioned far apart in the resulting small space.

Using the frequency of the function names called is the next step in selecting useful traces. A natural one dimensional representation of this data is the Shannon information [13], mathematically identical to the entropy of statistical mechanics. Other forms of entropy/information, obeying slightly less restrictive axiom lists, have been defined [14]. These extended entropies (as reviewed in [15]) are indexed by a parameter $q$ which, when $q = 1$ reduces them to the traditional Shannon entropy and which can be set to make them more ($q < 1$) or less ($q > 1$) sensitive to the frequency of infrequently called functions, improving the classification power of algorithms. Indeed, an extended Rényi entropy [16] with $q = 0$ returns the “number of unique function names” fingerprint, the Hartley entropy of information theory.

The entropy concept can also be extended in another way. Traces differ not only in which functions they call but in the pattern linking the call of one function with the call of another. As such it makes sense to collect not only the frequency of function calls, but also the frequency of calling given pairs, triplets, and in general $l$-tuples of calls. The frequency information assembled for these “$l$-words” can be converted into “word entropies”, for further discriminatory power. In addition each record in a trace can be encoded in different ways (denoted as $c$).

II. ENTROPIES AND TRACES: DEFINITIONS

In this section we describe techniques for extracting the probability of various events from traces (Section II A) and usage of this information to calculate entropies of traces (Section II B).

A. Extraction of probability of events from traces

A trace can be represented as a string, where each trace record is encoded by unique character. There exists a number of ways to encode the character. We concentrate on the following three character types $c$:

1. Record’s function name ($F$),
2. Record’s type ($FT$),
3. Record’s function names, type, and depth in the call tree ($FTD$).

In addition, we can generate consecutive and overlapping substrings $l$ of length $l$ from a string. We call such substrings $l$-words. For example, a string “ABCA” contains the following 2-words: “AB”, “BC”, and “CA”.

One can think of a trace as a message generated by a source with source dictionary $A = \{a_1, a_2, \ldots, a_n\}$ consisting of $n$ $l$-words $a_i$, and discrete probability distribution $P = \{p_1, p_2, \ldots, p_n\}$, where $p_i$ is probability of $a_i$. The dictionaries $A$ and their respective probability distributions $P$ for various values of $c$ and $l$ for the trace given in are shown in Table II.

Let us define a function $\alpha$ that, given a trace $t$, will return a discrete probability distribution $P$ for $l$-words of length $l$ and characters of type $c$: $P \leftarrow \alpha(t; l, c)$. (1)

The above empirical probability distribution $P$ can now be used to calculate entropy of a given trace for a specific $l$-word with characters of type $c$. We suppress the

---

2 The substring can start at any character $i$, where $i \leq n - l + 1$. 

---
dependence of the $P$ (and the individual $p_i$) on the $t$, $l$, and $c$. Let us now define entropies and discuss how we can utilize the $P$ in calculation of these entropies.

### B. Entropies and traces

The Shannon entropy $[13]$ is defined as

$$H_S(P) = -\sum_{i=1}^{n} p_i \log_b p_i,$$

where $P$ is the vector containing probabilities of the $n$ states, and $p_i$ is the probability of $i$-th state. Logarithm base $b$ controls the units of entropy. In this paper we set $b=2$, measuring entropy in bits.

Three extended entropies, Landsberg-Vedral $[17]$, Rényi $[16]$, and Tsallis $[18]$ are defined as:

$$H_L(P; q) = \frac{1 - 1/Q(P; q)}{1 - q},$$

$$H_R(P; q) = \frac{\log_2 [Q(P; q)]}{1 - q},$$

$$H_T(P; q) = \frac{Q(P; q) - 1}{1 - q},$$

where $q \geq 0$ is the entropy index, and

$$Q(P; q) = \sum_{i=1}^{n} p_i^q,$$

The extended entropies reduce to the Shannon entropy (by L’Hôpital’s rule) when $q = 1$. The extended entropies are more sensitive to states with small probability of occurrence than the Shannon entropy for $0 < q < 1$. Setting $q > 1$ leads to increased sensitivity of the extended entropies to states with high probability of occurrence.

The entropy $Z$ of a trace $t$ for a given $l$, $c$, and $q$ is calculated by inserting the output of Equation (1) into one of the entropies described in Equations (2) and (3):

$$Z \leftarrow H_E[\alpha(t; l, c); q],$$

where $E \in \{L, R, T, S\}$. Note that if $E = S$ then $q$ is ignored.

### III. Usage of Entropies for Classification of Traces

A typical scenario for trace comparison is the following. A software service analyst receives a phone call from a customer reporting software failure. The analyst needs to quickly determine the root cause of this failure and identify if 1) this is a rediscovery of a known defect exposed by some other customer in the past or 2) this is a newly discovered defect. If the first hypothesis is correct than the analyst will be able to quickly provide the customer with a fix-patch or describe a workaround for the problem. If the second hypothesis is correct the analyst must alert the maintenance team and start a full scale investigation to identify the root-cause of this new problem. In both cases time is of the essence — the faster the root cause is identified, the faster the customer will receive a fix to the problem and become satisfied.

In order to validate the first hypothesis, the analyst asks the customer to reproduce the problem with a trace capturing facility enabled. The analyst can then compare the newly collected trace against a library of existing traces collected in the past (with known root-causes of the problems) and identify potential candidates for rediscovery. To identify a set of traces related to similar functionality the library traces are usually filtered by names of functions present in the trace of interest. After that the filtered subset of the library traces is examined manually to identify common patterns with the trace of interest.

If the analyst finds existing trace with common patterns then the first hypothesis holds. Otherwise the analyst can conclude that this failure relates to a newly discovered defect and the second hypothesis is valid. With tens of thousands of traces in the library the manual approach becomes laborious. This process is similar in nature to usage of an Internet search engine. A user provides to the search engine keywords of interest and the engine’s algorithm returns a list of web pages ranked according to their relevance to keywords. The user examines the returned pages to identify pages most relevant to her.

---

3 This is a simplified description of the analysis process. In practice the analyst will examine defects with similar symptoms, consult with her peers, search database with description of existing problems, etc.
To automate this approach using entropies as fingerprints, we need an algorithm that would compare a trace against a set of traces, rank this set based on the relevance to a trace of interest, and then return the top X closest traces for manual examination to the analyst. In order to implement this algorithm we need a measure of distance between a pair of traces to quantify their closeness.

Formally, if \( D(t_i, t_j; M) \) satisfies three of the four usual conditions of a metric:

\[
\begin{align*}
D(t_i, t_j; M) &\geq 0, \\
D(t_i, t_j; M) &= D(t_j, t_i; M), \\
D(t_i, t_k; M) &\leq D(t_i, t_j; M) + D(t_k, t_j; M).
\end{align*}
\]

However, the fourth condition \( D(t_i, t_j; M) = 0 \iff t_i = t_j \) (identity of indiscernibles) holds true only for the fingerprints of traces; the actual traces may be different even if their entropies are the same. In other words, the identity of indiscernibles axiom only “half” holds: \( t_i = t_j \Rightarrow D(t_i, t_j; M) = 0 \), but \( D(t_i, t_j; M) = 0 \not\Rightarrow t_i = t_j \). As such, \( D \) represents a “pseudo-metric”. Note that \( D(t_i, t_j; M) \in [0, \infty) \) and our hypothesis is the following: the smaller the value of \( D \), the closer the traces.

Note that for a single pair of entropy-based fingerprints the normalization factor can be omitted and we define \( D \) as

\[
D(t_i, t_j; E , q, l, c) = |H_E[\alpha(t_i; l, c); q] - H_E[\alpha(t_j; l, c); q]|.
\]

We now define an algorithm for ranking a set of traces with respect to the trace of interest.

\[ D(t_i, t_j; M) = \sqrt{\sum_{k=1}^{m} \left( \frac{H_{E_k} [\alpha(t_i; l_k, c_k); q_k] - H_{E_k} [\alpha(t_j; l_k, c_k); q_k]}{\max \{H_{E_k} [\alpha(t; l_k, c_k); q_k]\}} \right)^2}, \]

where \( m \) is the number of elements in \( M \), and \( \max \{H_{E_k} [\alpha(t; l_k, c_k); q_k]\} \) denotes the maximum value of \( H_{E_k} \) for the complete set of traces under study for a given \( q_k \), \( l_k \), and \( c_k \). This denominator is used as a normalization factor to set equal weights to fingerprints related to different 4-tuples in \( M \).

\[ D(t_i, t_j; M) \geq 0, \]

\[ D(t_i, t_j; M) = D(t_j, t_i; M), \]

\[ D(t_i, t_k; M) \leq D(t_i, t_j; M) + D(t_k, t_j; M). \]

A drawback associated with usage of entropies as fingerprints is shown in Section III.D.

## A. Measure of distance between a pair of traces

We can obtain multiple entropy-based fingerprints for a trace by varying values of \( E, q, l \) and \( c \). Let us denote a complete set of 4-tuples of \( [E, q, l, c] \) as \( M \). We define the distance between a pair of traces \( t_i \) and \( t_j \) as:

1. Calculate distances between \( t \) and each trace in \( T \);
2. Order traces in \( T \) by their distance to trace \( t \) in ascending order;
3. Replace the vector of sorted traces with the vector of classes (e.g., defect IDs) to which these traces map;
4. Keep the first occurrence (i.e., the closest trace) of each class in the vector and remove the rest;
5. Calculate the ranking of classes taking into account ties using the “modified competition ranking” approach;
6. Return a list of classes with ranking smaller than or equal to \( X \).

The “modified competition ranking” can be interpreted as a worst case scenario approach. The ordering of traces of equal ranks is arbitrary; therefore we are looking at the case when the most relevant trace will always reside at the bottom of the returned list. To be conservative, we consider the outcome in which our method returns a trace in the top \( X \) positions as being in the \( X \)-th position.

Now consider an example of the algorithm:

1. Traces ranking algorithm: example

Suppose that we have five traces \( t_i, i = 1...5 \). The traces related to four software defects \( d_j, j = 1..4 \) as shown in Table II.

### B. Traces ranking algorithm

Given a task of identifying top \( X \) closest classes of traces from a set of traces \( T \) closest to trace \( t \) we resort to the following pseudo-algorithm:

1. Calculate distances between \( t \) and each trace in \( T \);
TABLE II. Example: Relation between traces and defects

<table>
<thead>
<tr>
<th>Defect</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>$t_5$</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$t_1, t_3$</td>
</tr>
<tr>
<td>$d_3$</td>
<td>$t_4$</td>
</tr>
<tr>
<td>$d_4$</td>
<td>$t_2$</td>
</tr>
</tbody>
</table>

Suppose that we calculate distances between traces using some measure of distance. The distances between trace $t$ and $t_{1..5}$ and defects’ ranks obtained using these hypothetical calculations are given in Table III. The traces are ranked based on the modified competition ranking schema. Trace $t_2$ is the closest to $t$, hence $d_4$ (to which $t_2$ is related) gets ranking number 1. Traces $t_1$ and $t_4$ have the same distance to $t$, therefore, $d_2$ and $d_3$ get the same rank. Based on the ranking schema algorithm we leave a gap before the set of items with the same rank and assign rank 3 to both classes. Traces $t_3$ and $t_5$ also have the same distance to $t$; however $t_3$ should be ignored since it relates to the already ranked defect $d_2$. This leads to assigning rank 4 to $d_1$. The resulting sets of top $X$ traces for different values of $X$ are shown in Table IV.

TABLE III. Example: Traces sorted by distance and ranked

<table>
<thead>
<tr>
<th>$t_i$</th>
<th>Distance between $t$ and $t_i$</th>
<th>Class (defect ID) of trace $t_i$</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>$d_4$</td>
<td>1</td>
</tr>
<tr>
<td>$t_1$</td>
<td>7</td>
<td>$d_2$</td>
<td>3</td>
</tr>
<tr>
<td>$t_4$</td>
<td>7</td>
<td>$d_3$</td>
<td>3</td>
</tr>
<tr>
<td>$t_3$</td>
<td>9</td>
<td>$d_2$</td>
<td>-</td>
</tr>
<tr>
<td>$t_5$</td>
<td>9</td>
<td>$d_1$</td>
<td>4</td>
</tr>
</tbody>
</table>

TABLE IV. Example: Top 1-4 defects

<table>
<thead>
<tr>
<th>Top $X$</th>
<th>Set of defects in Top $X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top 1</td>
<td>$d_4$</td>
</tr>
<tr>
<td>Top 2</td>
<td>$d_4$</td>
</tr>
<tr>
<td>Top 3</td>
<td>$d_4, d_2, d_3$</td>
</tr>
<tr>
<td>Top 4</td>
<td>$d_4, d_2, d_3, d_1$</td>
</tr>
</tbody>
</table>

C. Traces ranking algorithm: efficiency

The number of operations $C$ needed by the ranking algorithm is given by

$$C = c_1 O(|M||T|) + c_2 O(|T| \log |T|) + c_3 O(|T|)$$

where $c_1$ is a constant number of operations associated with $i$-th step, and $\cdot$ represents the number of elements in a given set. The coefficients $c_3, c_4$ and $c_5$ are of much smaller order than $c_1$ and hence terms corresponding to Steps 3, 4 and 5 do not contribute significantly to $C$. Pair-wise distance calculation, using (6), requires $O(|M|)$ operations. Therefore, calculation of distances between traces (Step 1) requires $O(|M||T|)$ operations. Assuming that $|M|$ remains constant, the number of operations grows linearly with $|T|$. The average sorting algorithm, required by Step 2 (sorting of traces by their distance to trace $t$), needs $O(|T| \log |T|)$ operations [21]. Usually, $c_1 \gg c_2$; this implies that a user may expect to see linear relation between $C$ and $|T|$ (even for large $|T|$), in spite of the loglinear complexity of the second term in (9).

The amount of storage needed for entropy-based fingerprints data (used by (9)) is proportional to

$$\phi|M||T| + \phi|M| = \phi|M|(|T| + 1),$$

where $\phi$ is the number of bytes needed to store a single fingerprint value. Term $a$ is the amount of storage needed for entropy-based fingerprints for all traces in $T$, and term $b$ is the amount of storage needed for the values of $\max \{H_{E_t}(\alpha(t; l_k, c_k); q_k)\}$ from (9). Assuming that $|M|$ remains constant, the data size grows linearly with $|T|$.

D. Entropies as fingerprints: drawback

The drawback associated with entropies comes from the fact that entropies cannot differentiate dictionaries of events, since entropy formulas operate only with probabilities of events. Therefore, entropies of strings “f1-f2-f3-f1” and “f4-f5-f6-f4” will be exactly the same for any value of $E$, $l$, $c$, and $q$. The simplest solution is to do a pre-filtering of traces in $T$ in the spirit of the SIFT framework described in Section I. For example, one can filter out all the traces that do not contain “characters” (e.g., function names) present in the trace of interest before using entropy-based fingerprints.
IV. VALIDATION CASE STUDY

We hypothesize that predictive classification power will vary with change in \( E, l, c, \) and \( q \). In order to study the classification power of \( H_E[\alpha(t; l, c); q] \) we will analyze Cartesian products of the following sets of variables:

1. \( E \in (S, L, R, T) \),
2. \( l \in (1, 2, \ldots, 7) \),
3. \( q \in (0, 10^{-5}, 10^{-4}, \ldots, 10^1, 10^2) \),
4. \( c \in (F, FT, FT D) \).

Let us denote the complete set of parameters obtained by the Cartesian product as \( \Lambda \).

Our software under study, called the Siemens suite, was first developed by Hutchins et al. [22] at the Siemens Corporate Research. It was further augmented and publicly made available at Software-artifact Infrastructure Repository [23, 24]. This software suite has been used by a large number of studies on defect analysis in the last decade (see [25, 26] for literature review).

The Siemens suite [22] contains seven programs. Each program has one original version and a number of faulty versions. A faulty version is a variant of the original version by one fault. A fault (changed source code from the original version) was seeded manually by Hutchins et al. [22]. A fault can span over multiple lines of source code and multiple functions. Each program comes with a collection of test cases, applicable to all faulty versions and the original program. A fault can be identified if the output of a test case on the original version differs from the output of the same test case on a faulty version of the program.

In this study, we experimented with the largest program “Replace” of the Siemens suite. It has 517 lines of code, 21 functions, 31 different faulty versions. There were 5542 test cases shared across all the versions. Out of these 31 \( \times \) 5542 test cases, 4266 (\( \approx 2.5\% \) of the total number of test cases) caused a program failure when exposed to the faulty program, i.e., were able to catch a defect. The remaining test cases were probably unrelated to the 31 defects. The traces for failed test cases were collected using a tool called Etrace [27]. The tool captures sequences of function-calls for a particular software execution such as the one shown in Figure 1. In other words, we collected 4266 function-call level failed traces for 31 faults (faulty versions) of the “Replace” program.

The distribution of the number of traces mapped to a particular defect (version) is given in Figure 2. Descriptive statistics of trace length are given in Table V. The length ranges between 11 and 101400 records per trace; average length is 623 records per trace. Average dictionary sizes for various values of \( c \) are given in Figure 3. Note that as \( l \) gets larger the dictionary sizes for all \( c \) start to converge.

| TABLE V. Descriptive statistics of length of traces |
| Min. | 1st Qu. | Median | Mean | 3rd Qu. | Max. |
| 11   | 218     | 380    | 623.3 | 678    | 101400 |

All of the traces contain at least one common function. Therefore, we skip the pre-filtering step. Note that direct comparison with existing trace comparison techniques is not possible since 1) the authors focus on identification of faulty functions [25, 26] instead of identification of defect IDs and 2) the authors [25] analyze a complete set of programs in the Siemens suite while we focus only on one program (Replace).

The case study is split into three parts: 1) analysis of the individual classification power of each \( H_E[\alpha(t; l, c); q] \) in Section IV A; 2) analysis of the classification power of the complete set of entropies in Section IV B.

A. Analysis of individual entropies

Analysis of the classification power of individual entropies is performed using 10-fold cross-validation. The validation process is designed as follows:

---

The “Replace” program had 32 faults, but the tool “Etrace” was unable to capture the traces of segmentation fault in one of the faulty versions of the “Replace” program. This problem was also reported by other researchers [22].
1. Randomly partition 4266 traces into 10 bins
2. For each set of parameters $E, l, c, q$
   a. For each bin
      i. Tag traces in a given bin as a validating set of data and traces in the remaining nine bins as a training set
      ii. For each trace $t$ in the validating set calculate the rank of $t$’s class (defect ID) in the training set using the algorithm in Section III B with (8) as the measure of distance and with the set of parameters $E, l, c, q$
   b. Average information about ranks of the “true” classes and store this data for further analysis

Our findings show that the best results are obtained for $H$ with $E \in \{L, R, T\}$, $l = 3$, $q \in (10^{-5}, 10^{-4})$, and $c = F DT$. Based on 10-fold cross validation, the entropies with these parameters were able to correctly classify $\approx 21.6\% \pm 1.1\%$ of Top 1 defects and $\approx 57.6\% \pm 1.5\%$ of Top 5 defects (see Table VI and Figure 4). Based on the standard deviation data in Table VI all six entropies show robust results. However, the results become slightly more volatile for high ranks (see Figure 5). Let us analyze these findings in details.

The $l$-words with $l = 3$ provide the best results based on the fraction of correctly classified traces in Top 5 (see Figure 4), suggesting that chains of three events provide optimal balance between the amount of information in a given $l$-word and the total number of words. As $l$ gets larger, the amount of data becomes insufficient to get a good estimate of the probabilities.

Comparison of three values of $c$ shows that $F DT$ outperforms $FD$ and $F$ (see Table VII). However, the difference between three values is marginal: for example $57.6\%$ in Top 5 for $c = F DT$ vs. $56.2\%$ for $c = F$. The fact that $F DT$ outperforms the remaining character types is expected, since $F DT$ contains the largest amount of information. However, the addition of information about type of trace point (entry or exit) does not significantly contribute to the classification power of the algorithm. Note that even though more time is needed to calculate the $F DT$-based entropies (since dictionary of $F DT$s will be twice as large as the dictionary of $FD$s), the comparison time remains the same (since the probabilities of $l$-words $P$ map to a scalar value via entropy function for all values of $c$).

---

6 Technically, in order to identify the true ranking one needs to tweak Step 6 of the algorithm and return a vector of 2-tuples [class, rank].

7 95% confidence interval, calculated as $\pm q(0.975, 9) / \sqrt{10} \times$ standard deviation, where $q(p, df)$ represents quantile function of the $t$-distribution, $p$ is the probability, and $df$ is degrees of freedom.
Our findings show that the extended entropies outperform the Shannon entropy for $q < 1$ and $q > 1$ (see Figure 4). However, performance of extended entropies with $q < 1$ is significantly better than with $q > 1$, suggesting that rare events are more important than frequent events for classification of defects in this dataset. The best results are obtained for $q = 10^{-4}$ and $q = 10^{-5}$.

It is interesting to note that classification performance is almost identical for $H_L$ with $E \in (L, R, T)$, $l=3$, $q \in (10^{-5}, 10^{-4})$, and $c = FDT$. We believe that this fact can be explained as follows: the key contribution to the ordering of similar traces (with similar dictionaries) for entropies with $q \rightarrow 0$ is affected mainly by a function of probabilities of traces’ events. This function is independent of $E$ and $q$ and depends only on $l$ and $c$, see Appendix A for details.

## B. Analysis of the complete set of entropies

Analysis of the classification power for the complete set of entropies is performed using 10-fold cross-validation in a similar manner to the process described in Section IV A.
FIG. 5. Fraction of correctly classified traces in Top 5 for $E = L$ and $c = FDT$. Solid line shows the average fraction of correctly classified traces in 10 folds; dotted line shows pointwise 95% confidence interval (95% CI) of the average.

However, instead of calculating distances for each $H$ independently, we now calculate distances between traces by utilizing values of $H$ for all parameter sets in $\Lambda$ simultaneously. The validation process is designed as follows

1. Randomly partition 4266 traces into 10 bins

   (a) For each bin
   
   i. Tag traces in a given bin as a validating set of data and traces in the remaining nine bins as a training set;
   
   ii. For each trace $t$ in the validating set calculate the rank of $t$’s class (defect ID) in the training set using the algorithm in Section III B with equation and all 4-tuples of parameters in $\Lambda$.

   (b) Average information about ranks of the “true” classes and store this data for further analysis.

The results shown in Table VII show the increase of predictive power: in the case of Top 1 the results improved from 21.6% (for individual entropies) to 29.7% (for all entropies combined); for Top 5 from 57.6% to 61.5%. A significant increase in computational effort (the number

9 We had to exclude a subset of entropies with $E = L$, $q = 10^2$ for all $l$ and $c$ from $\Lambda$. The values of entropies obtained with these parameters are very large ($> 10^{100}$), which leads to numeric instability of (6). We keep just one of the various named $q = 1$ entropies to avoid redundancy.
of entropy fingerprints increases from 1 to 504) does not yield dramatic improvement: the 7% increase in power for predicting Top 5 matches comes at a 503-fold increase in computational effort. We leave the resulting balance between cost and benefit for each individual analyst to make.

V. SUMMARY

In this work we analyze applicability of entropies to predictive classification of traces related to software defects. Our validating case study shows promising performance of extended entropies with emphasis on rare events \( q \in \{10^{-5}, 10^{-4}\} \). The events are based on triplets (3-words) of “characters” incorporating information about function name, depth of function call, and type of probe point \( (c = FDT) \).

In the future, we are planning to increase the number of datasets under study, derive additional measures of distance (e.g., using tree classification algorithms) and identify an optimal set of combinations of parameters.

Appendix A: Approximation of Equation (8)

We have observed that classification power of the \( H_E \) \( \alpha(t; l, c) \) \( q \) is the highest when \( q \rightarrow 0 \). In order to explain this phenomenon let us expand \( H_E \) \( \alpha(t; l, c) \) \( q \) using Taylor series:

\[
H_E [\alpha(t; l, c); q] \overset{q \rightarrow 0}{=} 1 - \frac{1}{n_i} + q \left( \frac{A_i}{n_i^2} + \frac{n_i - 1}{n_i} \right) + O(q^2),
\]

\[
H_R [\alpha(t; l, c); q] \overset{q \rightarrow 0}{=} \log_2(n_i) + q \left[ \frac{A_i}{\ln(2)n_i} + \log_2(n_i) \right] + O(q^2),
\]

\[
H_T [\alpha(t; l, c); q] \overset{q \rightarrow 0}{=} n_i - 1 + q (A_i + n - 1) + O(q^2),
\]

where \( A_i = \sum_{k=1}^{n_i} \ln(p_k) \). By plugging [A1] into [8] and assuming that for similar traces \( n \approx n_i \approx n_j \), [8] becomes:

\[
D(t_i, t_j; L, q, l, c) \approx \frac{q}{n^2} |A_i - A_j|, \tag{A2}
\]

\[
D(t_i, t_j; R, q, l, c) \approx \frac{q}{\ln(2)n} |A_i - A_j|, \tag{A2}
\]

\[
D(t_i, t_j; T, q, l, c) \approx q |A_i - A_j|,
\]

Equation [A2] can be interpreted as follows. In the case when \( q \rightarrow 0 \) and dictionaries of a pair of traces are similar, the key contribution to the measure of distance is coming from the \( \sum_{k=1}^{n_i} \ln(p_k) \) term (which depends only on \( l \) and \( c \)) making the rest of the variables irrelevant \( (q \) and \( n \) become parts of scaling factors). This can be highlighted by solving a system of equations to identify conditions that generate the same ordering for three traces \( t_i, t_j, t_k \) for all extended entropies (using approximations from [A2]):

\[
\begin{cases}
\frac{q}{n^2} |A_i - A_j| \leq \frac{q}{n^2} |A_i - A_k| \\
\frac{q}{\ln(2)n} |A_i - A_j| \leq \frac{q}{\ln(2)n} |A_i - A_k| \Rightarrow \\
q |A_i - A_j| \leq q |A_i - A_k| \Rightarrow |A_i - A_j| \leq |A_i - A_k|.
\end{cases}
\]

on AI Approaches to Fraud Detection and Risk Management, 50 (1997).


