Towards an Entropy-Based Analysis of Log Variability

Back, Christoffer Olling; Debois, Soren; Slaats, Tijs

Published in:
Business Process Management Workshops

DOI:
10.1007/978-3-319-74030-0_4

Publication date:
2018

Document Version
Peer reviewed version

Citation for published version (APA):
Back, C. O., Debois, S., & Slaats, T. (2018). Towards an Entropy-Based Analysis of Log Variability. In E. Teniente, & M. Weidlich (Eds.), Business Process Management Workshops: BPM 2017 International Workshops, Barcelona, Spain, September 10–11, 2017, Revised Papers (pp. 53-70). Springer. Lecture Notes in Business Information Processing, Vol.. 308 https://doi.org/10.1007/978-3-319-74030-0_4

Towards an Entropy-based Analysis of Log Variability

Back, Christoffer Olling; Slaats, Tijs; Debois, Søren

Published in:
Business Process Management Workshops

Publication date:
2017

Document Version
Peer reviewed version

Citation for published version (APA):
Back, C. O., Slaats, T., & Debois, S. (2017). Towards an Entropy-based Analysis of Log Variability. In Business Process Management Workshops Springer. (Lecture Notes in Business Information Processing).

Towards an Entropy-based Analysis of Log Variability

Christoffer Olling Back[1], Søren Debois[2], and Tijs Slaats[1] ⋆

[1] Department of Computer Science, University of Copenhagen
Emil Holms Kanal 6, 2300 Copenhagen S, Denmark
{back, slaats}@di.ku.dk

[2] Department of Computer Science, IT University of Copenhagen
Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
{debois}@itu.dk

Abstract. Process mining algorithms can be partitioned by the type of model that they output: *imperative* miners output flow-diagrams showing all possible paths through a process, whereas *declarative* miners output constraints showing the rules governing a process. For processes with great variability, the latter approach tends to provide better results, because using an imperative miner would lead to so-called ”spaghetti models” which attempt to show all possible paths and are impossible to read. However, studies have shown that one size does not fit all: many processes contain both structured and unstructured parts and therefore do not fit strictly in one category or the other. This has led to the recent introduction of hybrid miners, which aim to combine flow- and constraint-based models to provide the best possible representation of a log. In this paper we focus on a core question underlying the development of hybrid miners: given a (sub-)log, can we determine *a priori* whether the log is best suited for imperative or declarative mining? We propose using the concept of entropy, commonly used in information theory. We consider different measures for entropy that could be applied and show through experimentation on both synthetic and real-life logs that these entropy measures do indeed give insights into the complexity of the log and can act as an indicator of which mining paradigm should be used.

Keywords: Process Mining, Hybrid Models, Process Variability, Process Flexibility, Knowledge Work

1 Introduction

Two opposing lines of thought can be identified in the literature on process modelling notations. The *imperative* paradigm, including notations such as Petri nets [30] and BPMN [19], focuses on describing the flow of a process and is considered to be well-suited to structured processes with little variation. The *declarative paradigm*, including notations such as Declare [20], DCR Graphs [6], and GSM [14] focuses on describing

⋆ Authors listed alphabetically. This work is supported by the Hybrid Business Process Management Technologies project (DFF-6111-00337) funded by the Danish Council for Independent Research. We would like to thank both the reviewers and Jakob Grue Simonsen for their valuable and constructive feedback.

the rules of a process and is considered to be well-suited to unstructured processes with large degrees of variation. However, recent studies [7, 21] have shown that one size does not fit all: many processes do not fit strictly in one category or the other and instead contain both structured and unstructured parts. This has led to the recent introduction of a *hybrid* paradigm [21, 25], which aims to combine the strengths of these two approaches.

Following the introduction of the hybrid modelling paradigm, a number of hybrid mining algorithms have been developed: in [17] the authors use a heuristic approach based on the directly-follows-graph to divide activities between structured and unstructured parts of the model, in [26] the authors take a mixed approach and mine both a declarative and imperative model which are then overlain, and in [22] the authors take a model-based approach where first an imperative model is mined, which is analysed for pockets of unstructured behaviour, and for these pockets a declarative alternative is mined.

All these approaches avoid an important research question, first identified in [5]: *can we, based on an* a priori *analysis of the input log, measure if it is best suited to imperative or declarative mining?* Such a measure:

(i) Would give us greater insights into what type of miner we should use for a log;
(ii) could be applied to existing partitioning techniques [3, 10, 18, 27], determine for each partition if it is most suited to imperative of declarative mining, and hereby provide an efficient method to construct a hybrid model; and
(iii) could be used for the development of novel partitioning techniques that specifically aim to separate structured and unstructured behaviour in a log.

In this paper we propose basing such a measure on the notion of *entropy* from the field of information theory. Introduced by Shannon in his seminal 1948 paper [24], entropy measures the information content of a random variable. Intuitively, we can think of entropy as the "degree of surprise" we will experience when obtaining additional information about a system [2].

We propose that the entropy of an event log can serve as a predictor of whether the generating process is best modelled using declarative or imperative models. Highly structured processes should generate more homogeneous (low entropy) traces and more flexible processes should generate more varied (high entropy) traces. While information theoretic tools have been previously applied to predictive modelling, [**?**], our application to discriminating mining techniques is novel.

To find such a measure we first introduce a number of example logs that we use to illustrate our ideas and concepts in Section 2. In Section 3 we introduce three entropy measures on event logs: (i) trace entropy measures only the entropy on the level of distinct traces, (ii) prefix entropy measures entropy by taking into account all unique prefixes of the log, and (iii) block entropy measures entropy by considering all unique sub-strings present in the log. In Section 4 we report on an implementation of these measures and the results of applying them to both syntactic and real-life logs. We show that block entropy is the most successful measure, but suffers from a high computational complexity which becomes apparent on large logs with long traces. In addition it becomes clear that the current proposed measures are not yet absolute and that both further research and a more detailed evaluation are needed to arrive at such a measure. We discuss how we intend to do so in Section 5 and conclude in Section 6.

2 Running Example

We will use a running example of three logs to illustrate how we can use entropy to measure the variability of process logs. First let us recall the definitions of events, traces and logs.

Definition 2.1 (Events, Traces and Logs). *Let Σ be an alphabet of activities. An* event $e \in \Sigma$ *is a* specific occurrence *of an activity. A* trace $\sigma \in \Sigma^* = \langle e_1, \ldots, e_n \rangle$ *is a sequence of events $e_1, \ldots, e_n$, with each $e_i \in \Sigma$. Finally, a* log *is a multiset $[\sigma_1^{w_1}, \ldots, \sigma_n^{w_n}]$ where each $\sigma_i \in \Sigma^*$ and each $w_i \in \mathbb{N}$*

Notice that we have defined a trace as the sequence of activities observed in a particular process instance. A log is a multiset of such traces, representing explicitly the number of process instances exhibiting the particular trace.

Example 2.2. As a running example, consider the three logs L_1, L_2, and L_3 given in Figure 1. L_1 is a very structured log, for which we can easily find a compact imperative model, for example the Petri net shown in Figure 2. L_2 is the same log, except some traces are now more frequent than others. The last trace L_3 is a much less structured log, which results in a fairly complex imperative model, e.g. the Petri net in Figure 3, but can be be more effectively explained by a declarative model, as shown in Figure 4. The declarative model uses the Declare notation [20] and shows that: (i) a and b can not occur in the same trace, (ii) after an a we always eventually see an h, (iii) we must have seen at least one a before we can see a c, (iv) we must have seen at least one d before we can see a c, (v) we must have seen at least one d before we can see a e, (vi) after an e we always eventually see an f, (vii) we must have seen at least one f before we can see a g, (viii) after an f we will immediately see a g. One should note that in addition to giving a more straightforward view of the process, this model is also much more precise than the Petri net in Figure 3 (i.e. it allows less behaviour for which there is no evidence in the log).

3 Log entropy

Entropy is a measure of the information required to represent an outcome of a stochastic variable, intuitively indicating the "degree of surprise" upon learning a particular outcome [2]. For this paper we focus in particular on the notion of Shannon [24] entropy, which forms the foundation of the field of information theory.

Given a discrete random variable, X, taking on m possible values with associated probabilities $p_1, p_2, \ldots, p_m$, (Shannon) entropy, denoted H, is given by the expected value of the information content of X:

$$H = -\sum_{i=1}^{m} p_i \log_b p_i \tag{1}$$

L_1	L_2	L_3
$\langle a, b, c, d, f, g, h\rangle^5$	$\langle a, b, c, d, f, g, h\rangle^{15}$	$\langle h, a, h, d, c\rangle^5$
$\langle a, b, c, e, f, g, h\rangle^5$	$\langle a, b, c, e, f, g, h\rangle^8$	$\langle a, d, a, c, a, c, e, h, h, f, g\rangle^5$
$\langle a, b, c, d, f, g\rangle^5$	$\langle a, b, c, d, f, g\rangle^5$	$\langle d, e, h, f, g, e, f, g\rangle^5$
$\langle a, b, c, e, f, g\rangle^5$	$\langle a, b, c, e, f, g\rangle^2$	$\langle h, b, b, h, h\rangle^5$
$\langle a, b, b, c, d, f, g, h\rangle^5$	$\langle a, b, b, c, d, f, g, h\rangle^3$	$\langle b, h, d, b, e, h, e, d, f, g\rangle^5$
$\langle a, b, b, c, e, f, g, h\rangle^5$	$\langle a, b, b, c, e, f, g, h\rangle^4$	$\langle a, d, a, d, h\rangle^5$
$\langle a, b, b, c, d, f, g\rangle^5$	$\langle a, b, b, c, d, f, g\rangle^1$	$\langle b, f, g, d\rangle^5$
$\langle a, b, b, c, e, f, g\rangle^5$	$\langle a, b, b, c, e, f, g\rangle^2$	$\langle f, g, h, f, g, h, h, h\rangle^5$

Fig. 1. Example logs. L_1, L_2 are structured logs, differing only in number of occurrences of complete traces. L_3 is an unstructured log.

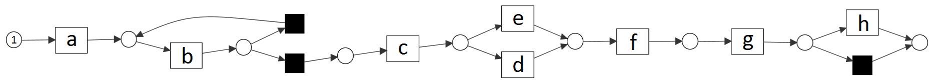

Fig. 2. Petri net for log L_1

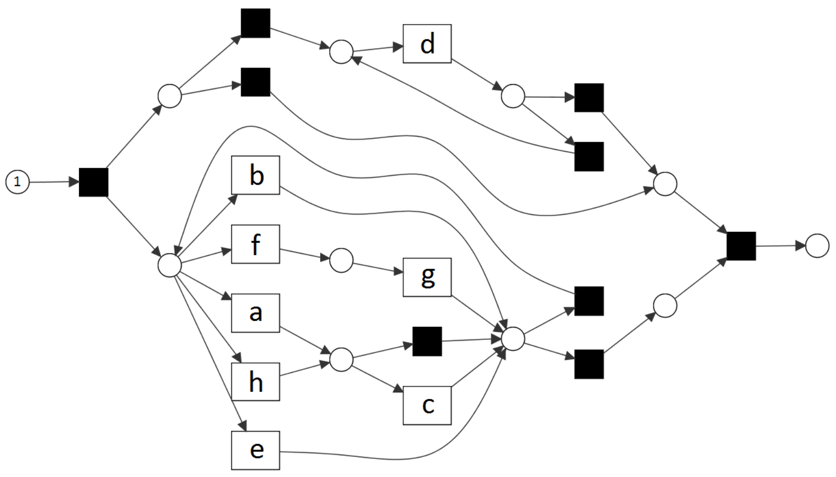

Fig. 3. Petri net for log L_3

Where b corresponds to the choice of coding scheme (i.e. for binary $b = 2$ and for decimal $b = 10$). We shall use the binary logarithm in the sequel.

Shannon justified this choice of measure with its 1) continuity w.r.t. p_i 2) monotonicity w.r.t. n under uniform distributions and 3) its weighted linearity under decomposition of choices[3].

The key question in using entropy as a measure of log complexity is *what would be the random variable implicit in a given log?*

[3] That is, $H(p_1, p_2, p_3) = H(p_1, (p_2 + p_3)) + (p_2 + p_3)H(p_2, p_3)$

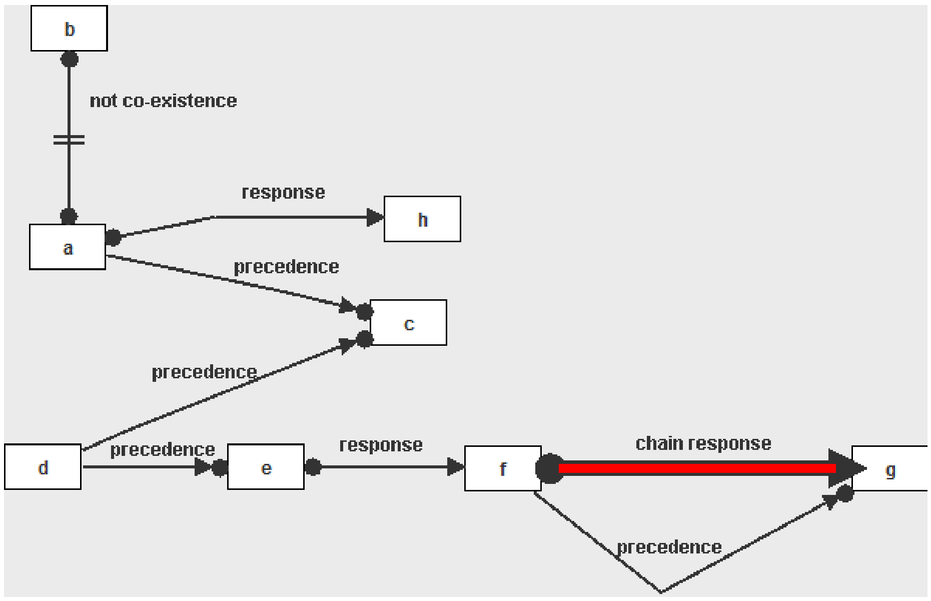

Fig. 4. Declare model for log L_3

3.1 Trace entropy

One very simple answer to this question is to take the underlying random variable as ranging over exactly the traces observed in the log, with probabilities exactly the frequencies observed in the log. This idea gives rise to the notion *trace entropy*.

Definition 3.1 (Trace entropy). *Let* $L = [\sigma_1^{w_1}, \ldots, \sigma_n^{w_n}]$ *be a log. The* trace entropy $t(L)$ *of* L *is the entropy of the random variable that takes the value* σ_i *with probability* $p_i = \frac{w_i}{\sum_{i=1}^{m} w_i}$.

Example 3.2. Even though the traces of L_1 and L_3 internally have radically different structure, they have the same number of occurrences of distinct traces, and so the same trace entropy:

$$t(L_1) = t(L_3) = -8 \times \frac{5}{40} \log_2 \frac{5}{40} = 3$$

Computing the trace entropy of L_2, we find

$$t(L_2) = -\left(\frac{15}{40} \log_2 \frac{15}{40} + \ldots + \frac{2}{40} \log_2 \frac{2}{40} \right) = 2.55$$

This example demonstrates that trace-entropy is likely *not* a good measure for determining if a model should be modelled imperatively or declaratively: L_1 and L_2 intuitively should mine to the same model, but have distinct trace-entropies. On the other hand, L_3 has much more variable behaviour than L_1, yet has the same trace entropy.

In general, if we are only interested in mining models with perfect fitness [4], then logs that differ only in the number of particular trace occurrences should not mine to different models. We are interested in the number of choices *available* at a particular point in a given trace, not the number of times a particular choice *was made* across all traces. We formalise this observation, using that in this simplistic setting, a "model" is really just a predicate on traces, or, if you like, a *language*.

Definition 3.3 (language equivalence). *Define logs L, L' to be language equivalent iff they are identical as sets, that is, for each $\sigma^w \in L$, there exists $\sigma^{w'} \in L'$ for some w', and vice versa.*

Lemma 3.4. *Let P be a predicate on traces; lift it to logs pointwise, ignoring multiplicity. Then if logs L, L' are language equivalent, we have $P(L)$ iff $P(L')$.*

Proof. Consider language equivalent logs $L = [\sigma_1^{w_1}, \dots, \sigma_n^{w_n}]$ and $L' = [\sigma_1^{w'_1}, \dots, \sigma_n^{w'_n}]$. By definition $P(L)$ iff $\forall i.P(\sigma_i)$ iff $P(L')$.

That is, taking the simultaneously abstract and simplistic view that mining a log L is tantamount to coming up with a predicate P s.t. $P(L)$, the above Lemma says that a mined model can never be used to distinguish language equivalent logs.

Because the output model cannot tell the difference between language equivalent logs, it would be unfortunate for our entropy measure to do so.

Definition 3.5. *An* entropy measure *is a function from logs to the reals. An entropy measure e* respects language equivalence *iff for any two language equivalent logs L, L', we have $e(L) = e(L')$.*

Trace entropy is unhelpful, then, because it does not respect language equivalence.

Example 3.6. The logs L_1, L_2 of Example 3.2 are language equivalent. However, they have different trace entropy measures. It follows that trace entropy does not respect language equivalence.

There is however on an intuitive level also a second reason that trace entropy is unhelpful: it does not consider the behaviour exhibited *within* the traces. We saw this in Example 3.2, where $t(L_1) = t(L_3)$; that is, trace entropy cannot distinguish internal structure of traces.

To consider the full behaviour of a log, we need to determine the entropy on the level of individual events.

3.2 Prefix entropy

To adequately measure the complexity of logs, we look inside individual traces. We must find a suitable notion of random variable that "generates" the traces we observe in the log, while at the same time characterises the internal structure of the individual traces.

Recall that a trace is the execution of a single process instance, taking the form of a sequence of events, or activity executions. At each point in a process execution, we will have a prefix of a completed trace. The distribution of these prefixes reflect to a certain extent the structure of the process.

Definition 3.7 (Prefix entropy). *Let L be a log. The* prefix entropy *of L, written $\epsilon(L)$ is defined as the entropy of the random variabockle which ranges over all prefixes of traces in L, and for each prefix $\langle e_1, \ldots, e_k \rangle \sqsubseteq \sigma \in L$ of a trace σ observed in a log L assigns as its probability the frequency of that prefix among all occurrences of prefixes in L.*

Example 3.8. In the log L_2, the prefix $\langle a, b, c, d \rangle$ occurs in 20 traces; the log contains a total of $15 \times 7 + 8 \times 7 + \ldots + 2 \times 7 = 280$ prefix occurrences, for a probability of $1/14$.

However, this notion of prefix entropy *does not* respect language equivalence, since logs differing only in the number of occurrences of a particular trace also differ in the set of occurrences of prefixes. Intuitively, we are interested in prefixes only as a measure of how much internal structure a log has, not how often various bits of that structure occurs. Hence, we disregard multiplicities of traces, in effect *flattening* the log.

Definition 3.9. *Let f be the function on logs which given a log L produces the corresponding set, i.e.,*

$$f([\sigma_1^{w_1}, \ldots, \sigma_n^{w_n}]) = [\sigma_1^1, \ldots, \sigma_n^1] = \{\sigma_1, \ldots, \sigma_n\}$$

The flattened prefix entropy *of L is $\epsilon \circ f(L)$, that is the prefix entropy applied to the flattened log.*

Example 3.10. In the log $f(L_2)$, the prefix $\langle a, b, c, d \rangle$ occurs just twice, among a total of only 56 prefix occurrences, for a probability of $1/26$.

We conjecture that transitioning from a log L to a flattened log $f(L)$ does not materially affect prefix entropy; we leave an investigation of exactly which properties are and are not preserved as future work.

Proposition 3.11. *The flattened prefix entropy $\epsilon \circ f$ respects language equivalence.*

Proof. Immediate by definition of ϵ.

Example 3.12. Computing the flattened event entropy of the example logs of Example 3.2, we find:

$$\begin{aligned} \epsilon \circ f(L_1) &= 4.09 = \epsilon \circ f(L_2) \\ \epsilon \circ f(L_3) &= 5.63 \end{aligned}$$

While the notion of flattened event entropy seems promising, there is one caveat. Because it is based on prefixes, it fails to appreciate structure that appears after distinct prefixes as such.

Example 3.13. Consider the log L_4 in Figure 5. This log is highly structured: it always contains exactly 4 activities; the first is a choice between $\{a, b, c, d, e\}$, the second an x, the third and fourth either x, y or y, x. (See Figure 6 for the Petri net displaying this behaviour.) However, this log has a trace entropy of $t(L_4) = 4.82$, much higher than the apparently *less* structured logs L_1 and L_2.

$$\begin{array}{c}
\langle a, x, y, z\rangle^5 \\
\langle a, x, z, y\rangle^5 \\
\langle b, x, y, z\rangle^5 \\
\langle b, x, z, y\rangle^5 \\
\langle c, x, y, z\rangle^5 \\
\langle c, x, z, y\rangle^5 \\
\langle d, x, y, z\rangle^5 \\
\langle d, x, z, y\rangle^5 \\
\langle e, x, y, z\rangle^5 \\
\langle e, x, z, y\rangle^5
\end{array}$$

Fig. 5. Log L_4 (highly structured).

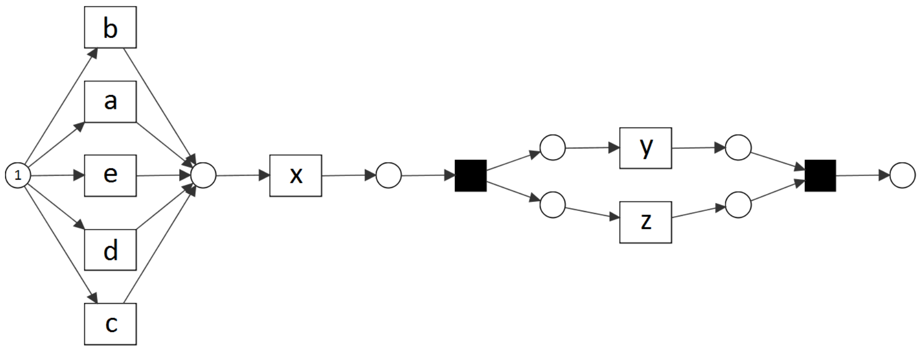

Fig. 6. Petri net for log L_4

3.3 Block entropy

To address the weaknesses of prefix entropy, we apply ideas from natural language processing [23], where entropy is studied in terms of n-length substrings known as n-grams.

We consider an individual trace a “word”, in which case our log is a multiset of such words, and look at the observed frequencies of arbitrary substrings within the entire log. That is, rather than looking at the frequencies of prefixes, we look at frequencies of substrings.

We shall see that while computationally more expensive, this idea alleviates the problems of prefix entropy and that observed structure is weighted equally, regardless of where it occurs in the log.

Definition 3.14 (k-block entropy). *Let L be a log. The k-*block entropy *of L, written $b_k(L)$ is defined as the entropy of the random variable which ranges over all k-length substrings of traces of L, assigning to each such substring s as probability the frequency of the number of occurrences of that substring among all occurrences of k-length substrings.*

Example 3.15. In the log L_4 in Figure 5, the 2-block $\langle x, y\rangle$ occurs 5 times; the log contains a total of $10 \times 3 = 30$ occurrences of 2-blocks, for a probability of $1/6$.

Following [23], we compute the k-block entropy $b_k(-)$ directly:

Lemma 3.16. *Let L be a log. The k-block entropy of L is given by*

$$b_k(L) = - \sum_{\langle s_1,\ldots,s_k\rangle \in \Sigma^\star} p(\langle s_1, \ldots, s_k\rangle) \log p(\langle s_1, \ldots, s_k\rangle)$$

Often in the literature on estimating the entropy of natural languages, text corpora are used in which all punctuation has been removed, meaning that sentences are ignored and blocks can cover the end of one sentence and beginning of another. For event logs we want to avoid finding spurious correlations among events at the end of one trace and beginning of another trace, so in our approach we keep a clear separation between traces.

We now define block entropy for all substrings up to the length of the longest trace. That is, instead of restricting the measure to blocks of length k, we include all blocks, from length 1 up to the length of the longest trace, in one entropy measure.

Definition 3.17 (All-block entropy). *Let L be a log. The* all-block entropy *of L, written $b(L)$ is the entropy of the random variable which ranges over all substrings of traces of L, assigning to each such substring s as probability the ratio of occurrences of that substring over all occurrences of substrings.*

Example 3.18. In the log L_3 in Figure 1, the substring (2-block) $\langle a, d\rangle$ occurs 3 times, once in the second entry, twice in the sixth. The log contains a total of 248 occurrences of substrings: $\Sigma_1^5 k = 15$ in the first trace, $\Sigma_1^{11} k = 66$ in the second, and so on. Altogether, the probability of $\langle a, d\rangle$ is $3/248$.

As for the prefix entropy, the all-block entropy does not respect language equivalence, but its flattening does.

Proposition 3.19. *The* flattened all-block entropy $b \circ f$ *respects language equivalence.*

Example 3.20. We give the flattened all-block entropies of the examples L_1 through L_4.

	L_1	L_2	L_3	L_4
$b \circ f(-)$	5.75	5.75	7.04	4.75

Notice how L_3 is still the highest-entropy log, but now L_4 is properly recognised as containing substantially less information than does L_1 and L_2.

We conclude this Section by noting that while the all-block entropy looks promising, it may be computationally infeasible to apply to large logs. Naively computing the all-block entropy of a log requires, in the worst case, tabulating the frequencies of all substrings seen in that log, an operation that takes polynomial space.

Assume a log has n traces, all of length k. A string of length k contains exactly $k - (i - 1)$ substrings of length i: one starting at each index except for the last $i - 1$ indices, where there is no longer room for a substring of length i.

Thus, by summing over all traces and all substring lengths, we can establish an upper bound on the size of the frequency tables for the substrings of a log:

$$n \times \sum_{i=0}^{k-1} k - i = \mathcal{O}(n \times k^2)$$

So in a concrete case where a log has 20.000 traces of length 10.000; we would in the worst case need a table of 2×10^{12} substrings. In the next section, we shall in one instance see our naive prototype implementation run out of memory on a somewhat smaller dataset.

4 Implementation and Early Experiments

To test the various measures we proposed we implemented a rudimentary ProM [31] plugin with support for computing the trace, prefix and block entropy of a given log. To get an indication of the utility of the entropy measures we applied them to both the examples L_1, L_2, L_3, and L_4 of the preceding sections, as well as to a selection of real-life logs. In particular we used the BPI Challenge 2012 [4], BPI Challenge 2013 (incidents) [5], hospital [6], sepsis cases [7], and road traffic fines [8] logs.

There is not yet a clear agreement in the literature on which of these logs should be mined imperatively, and which should be mined declaratively. However, it can be observed that the BPI Challenge 2012 log is commonly used as a base-case for declarative mining algorithms and that both the sepsis cases and hospital log result from highly flexible and knowledge-intensive processes within a Dutch hospital. A recent investigation involving the BPI Challenge 2013 (incidents) log seemed to indicate that an imperative approach may be the most successful, but no concrete conclusions were drawn [22].

We ran two sets of experiments: one to contrast the notions of trace, prefix and all-block entropy; and one to investigate more thoroughly the notion of k-block entropy.

4.1 Comparative measures

We report measures of trace, prefix and all-block entropies of the above-mentioned logs in Table 1.

The results are promising for the real-life logs we experimented on. In particular, we see that the BPI Challenge 2012 and Sepsis cases logs score highly in terms of all-block entropy, whereas the BPI Challenge 2013 log scores somewhat lower, which fits

[4] `https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f`

[5] `https://data.4tu.nl/repository/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07`

[6] `https://data.4tu.nl/repository/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`

[7] `https://data.4tu.nl/repository/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460`

[8] `https://data.4tu.nl/repository/uuid:270fd440-1057-4fb9-89a9-b699b47990f5`

Log	Event classes	Unique traces	Shortest trace	Longest trace	Entropy Trace	Entropy Prefix	Entropy All-block
L_1	8	8	6	8	3.00	4.09	5.75
L_2	8	8	6	8	2.55	4.09	5.75
L_3	8	8	4	11	3.00	5.63	7.04
L_4	8	10	4	4	3.32	4.82	4.75
BPI Challenge 2012	36	4366	3	175	7.75	12.53	16.01
BPI Challenge 2013	13	1511	1	123	6.66	11.32	12.21
Sepsis Cases	16	846	3	185	9.34	10.59	14.67
Road Traffic Fines	11	231	2	20	2.48	6.50	8.73
Hospital Log	624	981	1	1814	9.63	DNF	DNF

Table 1. Trace, flattened prefix and flattened all-block entropy measures for select logs.

the common intuition that the first two are most suited for declarative mining, whereas the latter is most suited to imperative mining. However, the difference in the results is relatively small and we will need to conduct further experiments on additional logs to determine what would make for a good cut-off value.

We were unable to compute the all-block entropy for the Hospital Log. This log has traces up to length 1800, and thus requires a large amount of memory to store the intermediary table of substring frequencies.

We conclude that (a) the flattened all-block entropy is the most promising measure for indicating whether a log is best mined imperatively or declaratively; and (b) that a computationally more efficient approximation to the all-block entropy is needed.

4.2 Block entropy measures

To understand the flattened block entropy measure in more detail, in particular in the hope of finding an efficient approximation of it, we analyse its constituent parts (1-blocks, 2-blocks etc.) in our selection of logs. The results are visualised in Figure 7.

We note that when blocks become longer than the longest trace, k-block entropy falls to zero since we are effectively counting the occurrences of impossible events and $lim_{n \to 0} n \log(n) = 0$. This contrasts with a system with one certain outcome in which case we also have $H = 0$ since $1 \log(1) = 0$. We emphasize that this plays no role in our all-block entropy measure since it includes only blocks up to the length of the longest trace.

Note that the entropy of L_3, a declarative process, is never less than those of the more structured logs, L_1, L_2, and L_4.

What is otherwise apparent from this figure is that there is no immediately obvious shortcut to the flattened all-block entropy obtainable as any particular k-block size: no single k seems representative of the full measure. This deficiency is further evident from the number of crossings in the diagram: it would appear that from no single k onwards does the entropies of individual logs maintain their relative positioning. E.g., at $k = 10$, the BPI 2013 log measures more complex than 2012; however, they meet and switch places at $k = 18$.

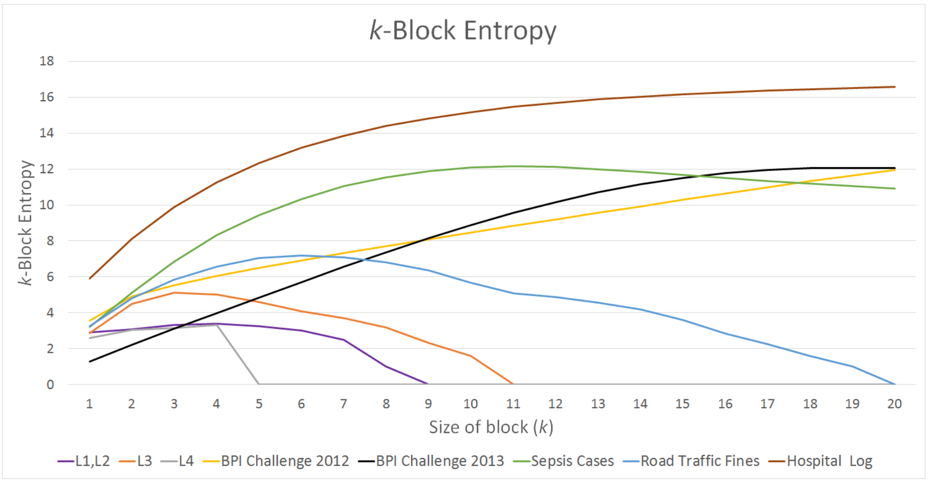

Fig. 7. k-block entropy of flattened logs using different block lengths.

5 Discussion and Future Work

Our experiments show that entropy is a promising indicator of log complexity. However, several questions are left open:

(i) How can we perform a more thorough evaluation of the suitability of our entropy measures?
(ii) Next to flattening the log, should we perform any additional normalisations to arrive at a fair measure of entropy?
(iii) Can we find entropy measures with a reasonable computational complexity so that we can deal with large logs?
(iv) How can we incorporate our approach with clustering techniques? (Both in an effort to find more efficient entropy estimations and use entropy measures to find suitable clusters for hybrid mining.)

In this section we shortly discuss these open challenges and provide possible avenues of future research to alleviate them.

5.1 More Thorough Experiments

In the previous section we reflected on the types of models we expected to be most suitable for the real-life logs that we experimented on. These were primarily educated guesses and to do a more thorough evaluation we should perform an analysis of these logs to determine whether they are most suited to imperative or declarative modelling.

One way to approach this could be to mine each log with imperative and declarative miners and compare the resulting models according to their precision and simplicity [4]. In addition it would be useful to experiment on a larger set of logs, including both a more comprehensive set of synthetic logs and additional real-life logs.

5.2 Additional Normalisation

In Section 3 we showed that it was useful to normalise a log by flattening it before determining its entropy. Other such normalisations might be necessary to arrive at a fair indication of the kind of miner that should be used. For example, our experiments show that larger logs result in a higher entropy measure, but it is questionable if a larger log is by definition always better suited to declarative modelling. Possible additional normalisations could be based on the number of unique activities in a log, the number of traces, the average length of traces, or the number of events.

5.3 Complexity of Entropy Measures

In certain cases, such as the hospital log, our proposed measure of flattened block entropy is computationally infeasible, at least for our naive implementation. Fortunately, the problem of approximating entropy is well-studied in the literature outside Computer Science, most notably in physics and natural language processing.

A more efficient approach is based on building prefix trees of non-overlapping blocks. One example of this is the Ziv-Lempel algorithm, which sequentially parses sequences into unique phrases, composed of their previously seen prefix plus a new symbol. In this way a tree structure is built with each phrase defined by a tuple representing a pointer to its prefix and the new symbol. Borrowing an example from [29], the string ABBABBABBBAABABAA would be parsed as:

A	B	BA	BB	AB	BBA	ABA	BAA
(0,A)	(0,B)	(2,A)	(2,B)	(1,B)	(4,A)	(5,A)	(3,A)

With the integers referring to the dictionary reference of earlier encountered prefixes (0 for root prefixes). It can be shown that the compression ratio of the Ziv-Lempel coding scheme converges to the entropy of an ergodic process [9] [29].

In [16], the authors found that on very short sequences, block entropies tended to lead to overestimates on low entropy sequences, while they outperformed Ziv-Lempel on high entropy sequences and suggest a two step process which uses a preliminary quick estimate of entropy to inform the choice of proper estimator.

5.4 Clustering of Logs

In determining whether the declarative or imperative modelling paradigm is most appropriate for a given event log, we may want to look more specifically at the similarity *between* traces rather than within traces. In other words, an event log consisting of nearly identical, but complex, traces may nonetheless be best modelled using an imperative approach, while a log of simple, but highly varied traces, may be best described by a declarative model. By clustering traces according to some distance metric, we can get an idea of the diversity of an event log.

[9] An ergodic process is one in which the statistical properties of the system are reflected in the observed data. Flipping one coin is an ergodic process, whereas flipping two coins, one fair, and one unfair is a non-ergodic process.

Most clustering techniques rely on Euclidean distance metrics, meaning that data must be represented as a d-dimensional vector. Even techniques such as expectation maximisation clustering, that don't rely directly on computing Euclidean distances, do assume that the data is independent and identically distributed. That is, that observed variables are not directly dependent on each other, so $p(a, b|c) = p(a|c)p(b|c)$. This is an issue for sequential data, a well-known problem in natural language processing, where the "bag-of-words" approach nonetheless leads to impressive results, for example in topic modelling and sentiment analysis [13]. In this approach, word order is simply ignored and documents are represented as multisets (a.k.a. bags) of words, which can then be represented as vectors, with word counts comprising the vector elements.

Similar approaches have been used for trace clustering, by representing traces as vectors of event occurrence counts, ignoring event ordering [11, 27]. For our purposes, this approach is not adequate. In trying to measure the entropy of event logs, event ordering cannot be ignored. The reason for this can clearly be seen from the interpretation of entropy as the amount of information gained upon learning the next symbol in a sequence *given* the preceding sequence. For example, in English the letter q is always followed by u, and so $p(u|q) = 1$ and therefore $h = H_n - H_{n-1} = -p(q, u)\log(u|q) = 0$.

To avoid the loss of ordering information which results from collapsing traces to vectors of event counts, we would need to find ways of estimating entropy which allow us to use non-Euclidean distance metrics, for example string edit distances [3, 8, 12]. This allows us to distinguish between traces consisting of the same (or similar) events, but in different orderings.

Notable clustering techniques that look promising includes Kozachenko and Leonenko's *nearest neighbour entropy estimator* [15].

One issue with k-nearest neighbours clustering, classification, and entropy estimation is that it is not entirely clear how to choose the optimal value of k. Previous research in trace clustering has dealt with this in part by using agglomerative hierarchical clustering techniques, which allow one to "zoom in" and "zoom out" on a hierarchy of clusters to find the optimal partitioning [28]. Furthermore, while we can use the above mentioned entropy estimators without actually converting traces into d dimensional vector representations, we do need to choose a value of d. Using a distance measure such as string edit distance, it is not clear what this value should be.

Correlation clustering is a method for grouping a set of objects into optimal clusters without specifying the number of clusters [1,9]. Just as important, correlation clustering is a graph-based approach, meaning that data is defined solely by edge weights between nodes, with edge weight representing the degree of similarity between nodes. For our current purposes, nodes would represent individual traces and edges the distance measure between them, for example string edit distance.

5.5 Dealing with Noise

In the current paper we don't address the issue of noise in process logs. Clearly noise is a source of variability and noisy logs will tend to have a large degree of entropy. The primary challenge is to distinguish between unintentional variability (noise) and intentional variability. One approach could be to first filter the log for noise using existing techniques, and then measure its entropy afterwards, accepting the risk of accidentally removing interesting behaviour from the log. Alternatively one could assume that the log contains no noise, measure its entropy, mine the log imperatively, declaratively, or hybridly based on the measure, and then analyse the resulting model for unintended flexibility.

6 Conclusion

In this paper we reported on an initial investigation of how entropy can be used as a measure for the complexity of a process log and thereby be used as a basis for determining if a log should be mined and modelled imperatively or declaratively. In particular we incrementally proposed three entropy measures, each building on the insights gained from the former. We arrived at a notion of block-entropy for process logs and showed through experiments on synthetic and real-life logs that this measure gives significant insights into the complexity of the log and accordingly which paradigm should be used for mining it. While the present experiments are positive, it is still too early for a definite conclusion on what makes for the *best* possible measure. Therefore we proposed 4 distinct paths along which the current work can be improved and we intend to follow-up on these suggestions in future work.

References

1. Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 238–247. IEEE, 2002.
2. Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer, 2006.
3. RP Jagadeesh Chandra Bose and Wil MP van der Aalst. Context aware trace clustering: Towards improving process mining results. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 401–412. SIAM, 2009.
4. J.C.A.M. Buijs, B.F. Dongen, and W.M.P. Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565, pages 305–322. Springer Berlin Heidelberg, 2012.
5. S. Debois and T. Slaats. The analysis of a real life declarative process. In *CIDM 2015*, pages 1374–1382, 2015.
6. Søren Debois, Thomas Hildebrandt, and Tijs Slaats. Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In *International Symposium on Formal Methods*, pages 143–160. Springer International Publishing, 2015.
7. Søren Debois, Thomas T. Hildebrandt, Morten Marquard, and Tijs Slaats. Hybrid process technologies in the financial sector. In *BPM '15 (Industry track)*, pages 107–119, 2015.
8. Pavlos Delias, Michael Doumpos, Evangelos Grigoroudis, and Nikolaos Matsatsinis. A non-compensatory approach for trace clustering. *International Transactions in Operational Research*, 2017.
9. Erik Demaine and Nicole Immorlica. Correlation clustering with partial information. *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, pages 71–80, 2003.
10. G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, Aug 2006.

11. Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.
12. Quang-Thuy Ha, Hong-Nhung Bui, and Tri-Thanh Nguyen. A trace clustering solution based on using the distance graph model. In *International Conference on Computational Collective Intelligence*, pages 313–322. Springer, 2016.
13. Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.
14. R. Hull, E. Damaggio, R. De Masellis, F. Fournier, M. Gupta, F.T. Heath, S. Hobson, M.H. Linehan, S. Maradugu, A. Nigam, P. Noi Sukaviriya, and R. Vaculín. Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *DEBS 2011*, pages 51–62, 2011.
15. LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.
16. Annick Lesne, Jean-Luc Blanc, and Laurent Pezard. Entropy estimation of very short symbolic sequences. *Physical Review E*, 79(4):046208, 2009.
17. Fabrizio Maria Maggi, Tijs Slaats, and Hajo A. Reijers. The automated discovery of hybrid processes. In *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings*, pages 392–399, 2014.
18. Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 1255–1264, New York, NY, USA, 2009. ACM.
19. Object Management Group. Business Process Modeling Notation Version 2.0. Technical report, Object Management Group Final Adopted Specification, 2011.
20. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. Declare: Full support for loosely-structured processes. In *EDOC 2007*, pages 287–300, 2007.
21. H.A. Reijers, T. Slaats, and C. Stahl. Declarative modeling-an academic dream or the future for bpm? In *BPM 2013*, pages 307–322, 2013.
22. Dennis M. M. Schunselaar, Tijs Slaats, Hajo A. Reijers, Fabrizio M. Maggi, and van der Aalst M.P. Wil. Mining hybrid models: A quest for better precision.
23. Thomas Schürmann and Peter Grassberger. Entropy estimation of symbol sequences. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 6(3):414–427, 1996.
24. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.
25. Tijs Slaats, Dennis M. M. Schunselaar, Fabrizio M. Maggi, and Hajo A. Reijers. The semantics of hybrid process models. In *CoopIS*, pages 531–551, 2016.
26. Johannes De Smedt, Jochen De Weerdt, and Jan Vanthienen. Fusion miner: Process discovery for mixed-paradigm models. *Decision Support Systems*, 77:123–136, 2015.
27. Minseok Song, Christian W Günther, and Wil MP Aalst. Trace clustering in process mining. In *Business Process Management Workshops*, pages 109–120. Springer, 2009.
28. Minseok Song, H Yang, Seyed Hossein Siadat, and Mykola Pechenizkiy. A comparative study of dimensionality reduction techniques to enhance trace clustering performances. *Expert Systems with Applications*, 40(9):3722–3737, 2013.
29. Joy A Thomas and Thomas M Cover. *Elements of information theory*. John Wiley and Sons, 2006.
30. W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 08:21–66, February 1998.
31. Wil M. P. van der Aalst, Boudewijn F. van Dongen, Christian W. Günther, Anne Rozinat, Eric Verbeek, and Ton Weijters. ProM: The process mining toolkit. In *BPM (Demos)*, 2009.