Using Minimum Description Length for Process Mining

T. Calders
TU Eindhoven
t.calders@tue.nl

C.W. Günther
TU Eindhoven
c.w.gunther@tue.nl

M. Pechenizkiy
TU Eindhoven
m.pechenizkiy@tue.nl

A. Rozinat
TU Eindhoven
a.rozinat@tue.nl

ABSTRACT

In the field of process mining, the goal is to automatically extract process models from event logs. Recently, many algorithms have been proposed for this task. For comparing these models, different quality measures have been proposed. Most of these measures, however, have several disadvantages; they are model-dependent, assume that the model that generated the log is known, or need negative examples of event sequences. In this paper we propose a new measure, based on the *minimal description length* principle, to evaluate the quality of process models that does not have these disadvantages. To illustrate the properties of the new measure we conduct experiments and discuss the trade-off between model complexity and compression.

1. INTRODUCTION

Process mining is aimed at the discovery of (business) processes within organizations. The events in these processes are recorded in logs that may include data generated by administrative services, health care or any other information system/workflow tool. Recently, several algorithms have been proposed to extract explicit models from event logs [7]. Given an event log, these algorithms try to find the most suitable model of some class of models that best fits the log. Different model classes that have been studied the problem of generating a model that describes the traces in a log or related problems include: Petri nets, Markov Models, grammar induction, etc. Over the last decade many process mining approaches have been proposed that are rooted in machine learning [3, 1].

In order to compare the quality of the discovered models, different measures have been proposed; e.g., soundness [2], behavioral and structural appropriateness [6], behavioral and structural precision and recall [4]. Most of them, however, have one or more of the following disadvantages: (a) They focus on one model class only. Measures such as the parsing measure, structural appropriateness, etc. are specifically designed towards Petri nets. (b) Some measures have a strong bias towards certain algorithmic techniques. (c) Some need negative examples ("forbidden" scenarios) as well as positive ones, which are often not available.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.
Copyright 2009 ACM 978-1-60558-166-8/09/03 ...$5.00.

This absence of a golden standard process quality measure that would allow researches to compare the performance of different process mining techniques is a major methodological problem. The major source of these problems is that process mining essentially is an unsupervised learning task. In contrast to, e.g., classification, there is no clear measurable task that needs to be learned, but rather the data needs to be described in a way that makes the information in it more accessible to the user. In clustering, similar problems are present. To alleviate the problem we propose a new measure based on the *minimal description length* (MDL) [5]. The MDL principle states that those models should be preferred that allow to describe the input data most succinctly.

To make the MDL principle work in the case of evaluating process models, we need to find a way to encode a log based on the model, and a way of encoding the model itself. In this paper[1], we show how this can be done for Petri net process models. The log encoding defined in this paper will be based on the enabled transitions in the replay of the log[2]. In this way, transitions that are enabled in a replay will have a much shorter encoding than faulty transitions, because for them, we will need to trigger an error recovery mechanism in the encoding (i.e., favoring models that are *correctly* describing the input data). Furthermore, models that have less transitions enabled during replay will have a shorter encoding than those allowing for many choices during replay (i.e., favoring models that are *accurately* describing the input data).

Besides having a criterion for model selection, someone may want to know how good (or bad) the current model is with respect to well-distinguishable reference points of the worst (or simply definitely bad) or an optimal (or simply definitely good) model. We will consider two reference models; one that optimizes log encoding but has high complexity (the explicit model), and one that has optimal model complexity but results in poor log encoding (the so-called flower model).

In the empirical evaluation we show the use of the new MDL evaluation criteria. Different process models will be evaluated on benchmark datasets showing the usefulness of our new evaluation criteria.

[1]An extended version is available as technical report, which can be found at *http://prom.win.tue.nl/research/wiki/mdl.*
[2]With 'log replay' we denote the process of firing transitions based on the events in the log.

ID	*Trace*
1	S, A, C, D, E
2	S, B, C, G, H, F, E
3	S, B, G, C, H, F, E

ID	*Trace*
4	S, B, H, C, F, E
5	S, B, C, H, F, E

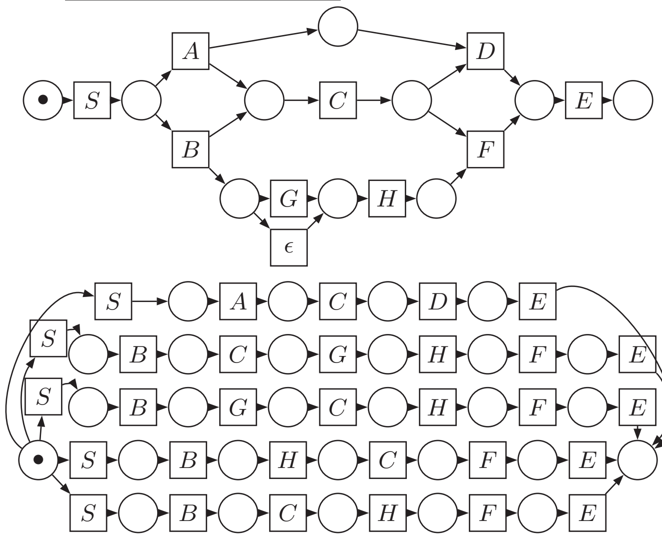


Figure 1: A log file and two petri nets that are able to replay the traces in the log.

2. PROBLEM STATEMENT

Before giving the problem statement, we quickly revise the most important notions of *Petri nets.* Petri nets are a modeling tool for the description of discrete processes, e.g., workflows. In Figure 1 two example Petri nets have been given. A Petri net consists of *transitions*, denoted by rectangles, *places*, denoted by circles, and directed arcs connecting places to transitions and vice versa. The transitions can have a label. The set of *input places* of a transition t, denoted $\bullet t$, consist of all places that have an outgoing arc to that transition, and the set of *output places*, denoted $t\bullet$ consists of those places that have an incoming arc from the transition. Any place can contain a non-negative number of *tokens*, denoted by black dots. A distribution of tokens over the Petri net is called a *marking*. A *marked* Petri net is a Petri net together with a marking. Formally, a marked Petri net will be described by $(\mathcal{P}(P, T, F, \lambda), M)$; P is the set of places, T the set of transitions, F the arc relation, and λ a function mapping transitions to their labels. A transition t not having a label will be denoted $\lambda(t) = \epsilon$. M is the marking of the net, mapping the places to a non-negative natural number.

A transition is *enabled* in a marked Petri net if all its input places have at least one token. *Firing* a transition results in a new marking, in which all input places of the transition have one token less than before firing, and all output places one more. A *trace* of a Petri net given an initial marking is the sequence of labels of a valid firing sequence of transitions; i.e., in the sequence of transitions, the first transition is enabled in the initial marking, the second transition is enabled in the marking resulting from firing the first transition, the third transition is enabled in the marking resulting from subsequently firing the first and the second transition, and so on. In Figure 1, all sequences in the log at the top are traces of both the marked Petri nets at the bottom.

Throughout this paper we will assume that all nets we deal with are such that every complete trace starts with S and ends with E, and S and E do not occur anywhere else in the trace. Notice that this requirement is not a limiting factor for our work; any marked Petri net can be transformed into such a Petri net. In the example nets in Figure 1, the traces in the log are all complete traces of both nets. Furthermore, $\langle S, B, G, H, C, F, E\rangle$ is a complete trace of the top net, but not of the bottom one.

The goal of process mining is, given a log, to generate a Petri net that describes as good as possible the workflow that generated the log. Note that the notion "describes as good as possible" is somewhat unclear and ambiguous to say the least. Recall, e.g., the log given in Figure 1; both nets are compatible with the log in the sense that all traces in the log are complete traces of the nets. The question which of the two nets is best representing the type of behavior observed in the log is not an easy one. The top net is somewhat more general, whereas the net at the bottom only allows for exactly those traces that are in the log.

The goal of this paper is to develop a new, well-founded measure for estimating the quality of a Petri net as a description of the workflow underlying the log. Our new measure is based on the MDL principle. The idea behind the measure is simple: on the one hand, if a Petri net captures a lot of the behavior of the log, it will be easy to compress the log using this Petri net. Suppose, for example that the log only contains valid and complete firing sequences of the Petri net. In that case we could opt to, instead of having to list all traces in the log completely, just list the firing sequence that resulted in this trace. As the number of enabled transitions is in general less than the number of different events, we will be able to encode this number more succinctly. On the other hand, the net itself should be sufficiently simple in order to enable for efficient log compression; it does not pay off if we get a huge compression of the log given a Petri net, but the net itself, which is needed for decoding and as such is part of the compression scheme, has excessive size. This trade-off between *log-compression* and *model complexity* is nicely illustrated in Figure 1; the bottom net will allow for more succinct log-compression as there is only one point of choice. The net itself, however, is quite extensive and hence the model complexity is high. Nevertheless, suppose that the frequency of every trace in the log of Figure 1 was 1000; i.e., the log consists of 1000 copies of the given log, it would be worth paying more model complexity in order to get huge log compression.

A second problem besides the trade-off between log compression and model complexity lies in the fact that most mined Petri nets are not fully conforming to the log. As such, we will need to take into account errors being made in the replay of the log. In the rest of the paper we will describe a new measure for estimating the quality of Petri nets that deals with these two problems.

3. MDL MEASURE

We consider a Petri net to be an accurate description of the log if the Petri net allows us to describe the log in a succinct way; that is: a good model should allow us to compress the log. To this end we first introduce an encoding of a log, relative to a Petri net. Then we will consider the encoding of the Petri net itself.

3.1 Encoding the Log

Let $(\mathcal{P}(P,T,F,\lambda),M)$ be a marked Petri net over $\mathcal{L}$ (a set of symbols), and let $L = \langle S, l_1, \ldots, l_n, E\rangle$ be a trace over $\mathcal{L}$. For the encoding we assume that the elements of $\mathcal{L}$ are ordered.

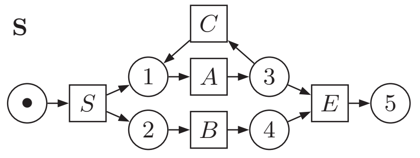
Instead of directly encoding L, we show how to encode a sequence of transitions $\sigma = \langle t_1, \ldots, t_m\rangle$. Notice that σ does not necessarily need to be a firing sequence of $(\mathcal{P}(P,T,F,\lambda), M)$. L is then encoded via a transition sequence σ with $\lambda(\sigma) = L$. When there is more than one σ with $\lambda(\sigma) = L$, we pick the one with the smallest encoding length. The encoding length of L w.r.t. the marked net $(\mathcal{P}(P,T,F,\lambda),M)$ is the minimum of the encoding lengths over all σ with $\lambda(\sigma) = L$.

Before we give the encoding, we first introduce some notations: M_0 will represent the initial marking M, and M_i, $i = 1 \ldots m$, the marking after the firing of $t_1, \ldots, t_i$. Sometimes a transition in the firing sequence is not enabled because the trace is not a valid firing sequence. P_i will denote the set of places in which a token needs to be inserted in M_{i-1} before t_i gets enabled. The resulting marking is denoted M'_i. E_i is the set of transitions enabled in M_i, and E'_i those that are enabled in M'_i.

Based on these notions, we introduce our encoding. If the trace is a firing sequence for the Petri net, basically, the encoding of the ith transition comes down to give its rank r_i in the list of enabled transitions E_{i-1}. In case of an error, however, the transition is not in this list. For this purpose, a special "rank 0" is introduced to denote an error, followed by the rank of the violating transition in the complete set of transitions T. This encoding scheme is illustrated in the following example:

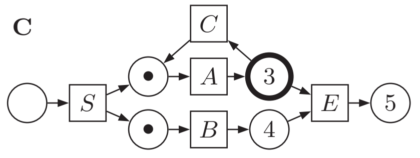
Consider the following Petri net and the coding of the trace $\langle S,C,A,C,A,E\rangle$ that is $\langle 1,\ 0,4,\ 1,\ 3,\ 1,\ 0,5\rangle$.

S



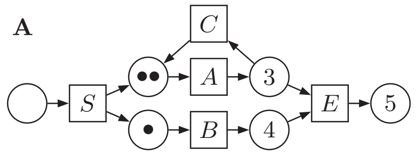
$$M_0 = \{0\} \quad E_0 = \{S\} \quad P_0 = \{\} \quad M'_0 = \{0\} \quad r_1 = 1$$

C



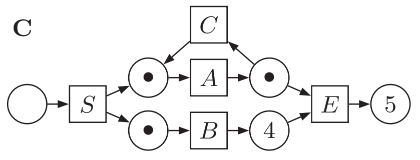
$$M_1 = \{1,2\} \quad E_1 = \{A,B\} \quad P_1 = \{3\} \quad M'_1 = \{1,2,3\} \quad r_2 = 0$$

A



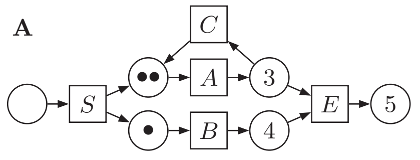
$$M_2 = \{1,1,2\} \quad E_2 = \{A,B\} \quad P_2 = \{\} \quad M'_2 = \{1,2\} \quad r_3 = 1$$

C



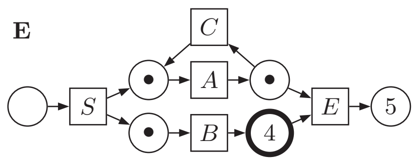
$$M_3 = \{1,2,3\} \quad E_3 = \{A,B,C\} \quad P_3 = \{\} \quad M'_3 = \{1,2,3\} \quad r_4 = 3$$

A



$$M_4 = \{1,1,2\} \quad E_4 = \{A,B\} \quad P_4 = \{\} \quad M'_4 = \{1,1,2\} \quad r_5 = 1$$

E



$$M_5 = \{1,2,3\} \quad E_5 = \{A,B,C\} \quad P_5 = \{4\} \quad M'_5 = \{1,2,3,4\} \quad r_6 = 0$$

For reasons of clarity, extra spacing is inserted between the encodings of the different transitions. The first integer 1 indicates that the first (and only) enabled transition is chosen; S. The second transition is encoded by the error indicator 0, and the rank 4 of the transition C in the set of all transitions T. The next 3 transitions are valid and have respectively ranks 1, 3, and 1 in their sets of enabled transitions. The last transition is again invalid, so the code 0 is used, and the rank of E in T, 5, is given.

3.2 Encoding of the Model

The assumption of dedicated start and end symbols S and E is reflected by the presence of the transitions t_s and t_e and the places p_s and p_e. $\lambda(t_s) = S$, $\lambda(t_e) = E$, and the place t_s has only one outgoing arc to t_s, p_e has only one incoming arc from t_e. The initial marking will always be one token in place p_s.

Let $\mathcal{P}(P,T,F,\lambda)$ be a Petri net, $P = \{p_s, p_e, p_1, \ldots, p_l\}$, $T = \{t_s, t_e, t_1, \ldots, t_k\}$. For reasons of simplicity, we assume that if a Petri net has a transition with label l_i, then it has also at least one transition with label l_j, for all $0 < j < i$, where l_i denotes the element with rank i in $\mathcal{L}$. This requirement is not really a restriction as we can always reorder the set of symbols to meet it. Let $\lambda(t_i)$ be the rank of the label of transition t_i. The encoding of $\mathcal{P}(P,T,F,\lambda)$ is the following:

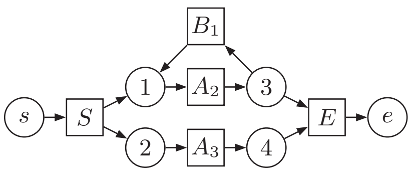
$$\langle k\rangle \cdot \langle \lambda(t_1), \lambda(t_2), \ldots, \lambda(t_k)\rangle \cdot enc(p_1) \cdot \ldots \cdot enc(p_l)\rangle \ ,$$

where $enc(p)$ denotes the following encoding of the place p: let $\bullet p \setminus \{t_s\} = \{t_{i_1}, \ldots, t_{i_p}\}$, and $p \bullet \setminus \{t_e\} = \{t_{o_1}, \ldots, t_{o_q}\}$; (i.e., apart from potential connections to t_s and t_e, place p has incoming arcs from transitions $t_{i_1}, \ldots, t_{i_p}$ and outgoing arcs to $t_{o_1}, \ldots, t_{o_q}$). Let $Num(\bullet p)$ now denote $\{i_1, \ldots, i_p\} \cup \{0 \mid p_s \in \bullet p\}$ and $Num(p\bullet)$ now denote $\{o_1, \ldots, o_p\} \cup \{0 \mid p_e \in p\bullet\}$ That is, for the special transitions t_s and t_e, the number 0 is used; as t_s cannot appear in the output list of a place, and t_e not in the input list, using 0 to encode both does not lead to ambiguities. We then have:

$$enc(p) = \langle |\bullet p|, |p\bullet|, i_1, i_2, \ldots, i_p, o_1, o_2, \ldots, o_q\rangle$$

For example, a place with inputs from transitions t_s, t_1, t_2 and output to t_e, t_4 is encoded as $\langle 3, 2,\ 0, 1, 2,\ 0, 4\rangle$. Notice that the places p_s and p_e are not encoded in the model, as their connections are fully known in advance.

Consider the following Petri net:



The order between the transitions is B_1, A_2, A_3; i.e., $t_1 = B_1$, $t_2 = A_2$, $t_3 = A_3$. The labels are: $\lambda(A_2) = A$, $\lambda(A_3) = A$, $\lambda(B_1) = B$, and the order on the labels is the alphabetic order. The order of the places is indicated by their number.

The encoding of this model is as follows:

$$\langle 3\rangle \cdot \langle 2,1,1\rangle \cdot \langle 2,1,0,1,2\rangle \cdot \langle 1,1,0,3\rangle \cdot \langle 1,2,2,1,0\rangle \cdot \langle 1,1,3,0\rangle$$

The encoding has been split into parts to increase the readability; the leading 3 indicates that, besides the begin- and end-transitions and -places, there are 3 transitions. The next block $2,1,1$ indicates that the labels of the three additional transitions are respectively $l_2 = B$, $l_1 = A$, and again $l_1 = A$. After that, the encodings of the 4 places $p_1, \ldots, p_4$ follow. For the first place, the encoding $2,1,0,1,2$ indicates that this place has two incoming arcs, and one outgoing arc (leading $2,1$). The incoming arcs come from transitions t_s and t_1 $(0,1)$, and the outgoing arc goes to transition t_2 (2). The complete encoding is the concatenation of all these elements.

3.3 Log and Model Encoding Length

To encode an element of a predefined set with b elements, $\log_2(b)$ bits are needed (binary representation). When there are k violating transitions, the encoding cost of a log with n events is hence:

$$\left\lceil \sum_{i=1}^{n} \log_2(|E_{i-1}|+1) + k\log_2(|T|) \right\rceil$$

Indeed; for every transition t_i, either the value 0 if it is invalid, or its rank in the set E_{i-1} has to be given. There are hence $|E_{i-1}|+1$ values between which there has to be chosen, and hence $\log_2(|E_{i-1}|+1)$ bits are needed. For the violating transitions, besides the error code 0, also $\log_2(|T|)$ bits have to be given to indicate the rank of the transition in the set T.

We will use similar techniques for determining the length of the model encoding. Encoding a number requires $\log_2(n)$ bits if the number is in the range $1\ldots n$, and $\log_2(n+1)$ if the number is in the range $0\ldots n$, and for a number b in the range $0\ldots$; i.e., on which no upper bound is known requires $2\lceil\log_2(b+1)\rceil+1$ bits. This encoding of an arbitrary number is as follows: let $b_1 \ldots b_k$ be the binary representation of i. The encoding is then:

$$\overbrace{1\ldots1}^{k\times}\ 0\ b_1 b_2 \ldots b_k$$

The k leading 1's are used to indicate the length of the binary encoding. The first 0 marks the boundary between the length indicator and the actual binary representation.

Hence, the length of the encoding

$$\langle k\rangle \cdot \langle \lambda(t_1),\ldots,\lambda(t_k)\rangle \cdot enc(p_1)\cdot \ldots \cdot enc(p_l)\rangle \text{ is}$$

$$\lceil (2\lceil\log_2(k)\rceil+1) + k\log_2(k+1) + \sum_{i=1}^{l} length(enc(p_i))\rceil\ ,$$

where $length(enc(p))$ for $enc(p) = \langle |\bullet p|, |p\bullet|, i_1, i_2, \ldots, i_p, o_1, o_2, \ldots, o_q\rangle$ equals $2\log_2(k+1) + (|\bullet p| + |p\bullet|)\log_2(k+1)$. We can reorder the terms in this sum; every arc either starts in a place, or ends in one, and hence adds exactly $\log_2(k+1)$ bits to the sum. So, if n_a denotes the total number of arcs, excluding (p_s, t_s) and (t_e, p_e), we get the following encoding length for the model:

$$\lceil 2(\lceil\log_2(k+1)\rceil) + 1 + k\log_2(k+1) + 2l\log_2(k+1) + n_a\log_2(k+1)\rceil\ .$$

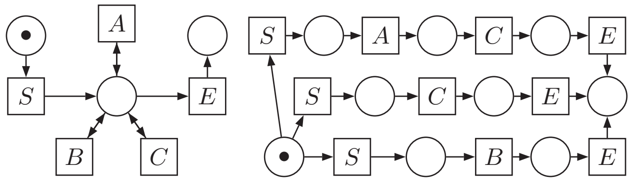


Figure 2: FRM (left) and ERM (right) for a log D with the traces $\langle S,A,C,E\rangle$, $\langle S,C,E\rangle$, and $\langle S,B,E\rangle$.

4. REFERENCE MODELS

An MDL-based approach to guide the model selection process is rather straightforward. The main intuition of deciding if the current model is the best among the computed alternatives is as simple as counting, for each model, the numbers of bits that are required to encode all the traces in the log and the number of bits needed to encode the corresponding models. The best one is that model that corresponds to the least number of bits. That is, given a set of Petri net models, the best model M is the one that minimizes $L(M) + L(D|M)$, where $L(M)$ is the length, in bits, of the model encoding, and $L(D|M)$ is the length, in bits, of the data D (event log) encoded with M.

However, besides having such criterion for model selection, an analist looking for a good model typically also wants to know what the progress of the learning process is, i.e., how good or how bad the current model is with respect to well-distinguishable reference points. These reference points correspond to the worst (or simply definitely bad) models and the optimal (or simply definitely good) models for the different criteria.

The estimation of the encoding costs of a log consists of two parts; the cost of encoding the log with a given model and the cost of encoding the model itself. Therefore, we need to consider two corresponding baselines and to show how they relate to each other. One extreme situation is when we model a log with a Petri net that allows for the execution of the given activities (the labels in the set of symbols) in any order. Such a model is shown in Figure 2 (left). We will refer to this model as the *flower reference model* (FRM).

It can be shown that given an event log and a set of possible actions/states, the cost of $L(FRM)$ is minimal (proportional to the logarithm of the number of places). The cost $L(D|FRM)$ of encoding the event log with the FRM, however, is very high, as every transition, except for the transition labeled S, is always enabled.

Another extreme situation is when we model a log with a Petri net that explicitly includes every trace in the log as part of the model. We will refer to such a model as to an *explicit reference model* (ERM). It can be shown that the cost of $L(D|ERM)$ is minimal since we only need to distinguish traces from each other, but the $L(ERM)$ model encoding costs are high. The ERM is illustrated in Figure 2 (right).

These two extreme cases, the FRM and the ERM provide us with a nice possibility of normalizing and visualizing the encoding costs of a model within the *log compression* (lc) and the *model simplicity* (ms) dimensions as shown in Figure 3. The performance of each process mining technique can be mapped to one point and compared. It is worth notic-

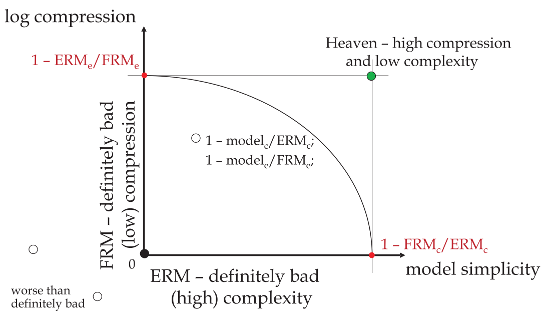

log compression
Heaven – high compression and low complexity
1 – ERM_e/FRM_e
FRM – definitely bad (low) compression
1 – model_c/ERM_c; 1 – model_e/FRM_e;
1 – FRM_c/ERM_c
model simplicity
0
ERM – definitely bad (high) complexity
worse than definitely bad

Figure 3: Complexity/compression trade-off.

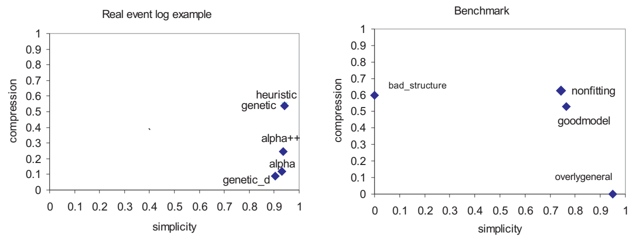


Figure 4: Experimental results.

ing that in general ms and lc are unbounded in the sense that we can always induce more and more complex process models. Eventually, this might result in having negative scores in the plot. However, we may prefer either neglecting models that are worse than the definitely bad models or rounding the negative values to zero.

A parameter α can be introduced to specify the relative importance of log compression and model simplicity for the analyst. A weighed total cost $I = \alpha * lc + (1 - \alpha) * ms$ would reflect a preference with respect to this trade-off. Thus, our MDL-based measure can be used to guide a "balanced" model selection or favoring Petri nets of the "desired" complexity to compression ratio by choosing an appropriate α value.

5. EXPERIMENTAL RESULTS

Figure 4 illustrates how different process mining techniques (or similar techniques with different parameter settings) can be compared. Each point in the plot on the left corresponds to the MDL-based quality measures of the process model obtained with the particular process mining technique (Alpha miner, Alpha++ miner, Heuristic miner, and (Duplicates) Genetic miner, all of which are available in *ProM* 5.0 [3]) on a real event log.

We can see that the Heuristic miner and Genetic miner, being more robust to noise [4], can produce more general and less overfitting models. They achieve the best compression results in comparison with the Alpha and Alpha++ miners and the Duplicates Genetic miner (genetic_d). The Duplicates Genetic miner supports the modeling of processes with nets with duplicate tasks; that is, different transitions can have the same label. It clearly has the worst performance on this event log due to a poor compression rate (an order of magnitude lower than with the Heuristic and the Genetic miner).

The plot on the right corresponds to some manually created models for the artificially generated benchmark event log. We can see from this benchmark example that, as expected, the *bad_structure*, which is very close to the ERM, although having reasonable compression ratio, scores very poorly with respect to model simplicity. The *overlygeneral* model, on the contrary, is much closer to the FRM. *nonfitting* and *goodmodel* are surprisingly close to each other and illustrate that gaining a little in simplicity, the *goodmodel* in fact loses in log compression.

[3]The latest ProM is freely available at http://prom.sf.net/.

6. CONCLUSION

Evaluation of process models and corresponding process mining techniques and process modeling languages that are used to extract and express these models from the event logs is a non-trivial task. There is no golden standard currently available, and existing measures for assessing the quality of process models have certain limitations that result in biases and subjectiveness in the evaluation and comparison of different process modeling languages and techniques. In this paper we introduced MDL-based process model quality measure that is objective and can be used for the assessment of the appropriateness of different modeling languages and process mining algorithms with respect to the compactness and fitness of the produced models. This potential of bridging the gap between different process modeling languages can not be underestimated. In this paper we demonstrated how an MDL-based quality measure can be defined for the Petri net modeling language. We plan to introduce the same principle for other languages and formalisms as well.

Our future work includes the application of our MDL-based process quality measure for guiding process mining techniques, i.e., favoring the construction of models with the "desired" complexity to compression ratio.

7. REFERENCES

[1] G. Greco, A. Guzzo, G. Manco, and D. Sacca. Mining unconnected patterns in workflows. *Inf. Syst.*, 32(5):685–712, 2007.

[2] G. Greco, A. Guzzo, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Trans. on Knowl. and Data Eng.*, 18(8):1010–1027, 2006.

[3] J. Herbst. A machine learning approach to workflow management. In *Proc. of ECML'00*, pages 183–194, London, UK, 2000. Springer-Verlag.

[4] A. K. Medeiros, A. J. Weijters, and W. M. Aalst. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304, 2007.

[5] J. Rissanen. Modelling by the shortest data description. *Automatica*, 14:465–471, 1978.

[6] A. Rozinat and W. M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.

[7] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, 2004.