Process Equivalence in the Context of Genetic Mining

W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{w.m.p.v.d.aalst,a.k.medeiros,a.j.m.m.weijters}@tm.tue.nl

Abstract. In various application domains there is a desire to compare process models, e.g., to relate an organization-specific process model to a reference model, to find a web service matching some desired service description, or to compare some normative process model with a process model discovered using process mining techniques. Although many researchers have worked on different notions of equivalence (e.g., trace equivalence, bisimulation, branching bisimulation, etc.), most of the existing notions are not very useful in this context. First of all, most equivalence notions result in a binary answer (i.e., two processes are equivalent or not). This is not very helpful, because, in real-life applications, one needs to differentiate between slightly different models and completely different models. Second, not all parts of a process model are equally important. There may be parts of the process model that are rarely activated (i.e., "process veins") while other parts are executed for most process instances (i.e., the "process arteries"). Clearly, differences in some veins of a process are less important than differences in the main artery of a process. To address the problem, this paper proposes a completely new way of comparing process models. Rather than directly comparing two models, the process models are compared with respect to some typical behavior. This way, we are able to avoid the two problems just mentioned. The approach has been implemented and has been used in the context of *genetic process mining*. Although the results are presented in the context of Petri nets, the approach can be applied to any process modeling language with executable semantics.

Keywords: Process Mining, Petri Nets, Genetic Algorithms, Process Discovery, Business Process Intelligence, Process Equivalence.

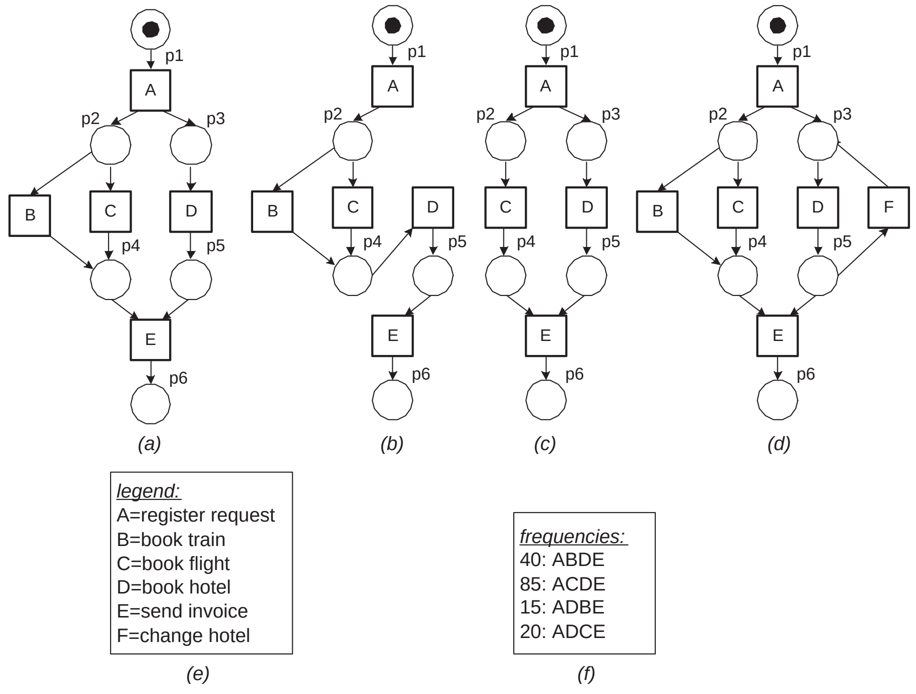
1 Introduction

Today one can find a wide variety of process models in any large organization [16]. Typical examples are:

- reference models (e.g., the EPC models in the SAP R/3 reference model [22])
- workflow models (e.g., models used for enactment in systems like Staffware, FLOWer, FileNet, Oracle BPEL, etc. [2]),
- business process models/simulation models (e.g., using tools such as ARIS, Protos, Arena, etc. [16]),

– interface/service descriptions (e.g., the Partner Interface Processes in RosettaNet [28], the abstract BPEL processes in the context of web services [8], choreography descriptions using WSCDL [21], or other ad-hoc notations [31]), or
– process models discovered using process mining techniques [5, 6].

Given the co-existence of different models and different types of models, it is interesting to be able to compare process models. This applies to different levels ranging from models at the business level to models at the level of software components (e.g., when looking for a software component matching some specification). To compare process models in a meaningful manner, we need to assume that these models have semantics. Moreover, we need to assume some equivalence notion (When are two models the same?) People working on formal methods have proposed a wide variety of equivalence notions [1, 17, 24], e.g., two models may be identical under trace equivalence but are different when considering stronger notions of equivalence (e.g., bisimulation). Unfortunately, most equivalence notions provide a "true/false" answer. In reality there will seldom be a perfect fit. Hence, we are interested in the *degree of similarity*, e.g., a number between 0 (completely different) and 1 (identical). In other to do so, we need to quantify the differences. Here it seems reasonable to put more emphasis on the frequently used parts of the model.



legend:
A=register request
B=book train
C=book flight
D=book hotel
E=send invoice
F=change hotel

frequencies:
40: ABDE
85: ACDE
15: ADBE
20: ADCE

Fig. 1. Running example.

To clarify the problem, let us consider Figure 1 where four process models (expressed in terms of Petri nets [13, 20, 27]) are depicted. These models describe the booking of a trip, see the legend for the interpretation of the various transitions in the Petri nets, e.g., C refers to the booking of a flight. Clearly, these models are similar. However, using classical equivalence notions all models are considered different. For example, in process (a) it is possible to have the execution sequence $ADBE$ while this sequence is not possible in (b) and (c). Moreover, the Petri net in Figure 1(d) allows for $ACDFDE$ which is not possible in any of the other models. Note that we focus on the active parts of the net (i.e., the transitions) rather than passive things such as places. Although classical equivalence notions consider the four models to be different, it is clear that some are more similar than other. Therefore, we want to quantify "equality", i.e., the degree of similarity. A naive approach could be to simply compare the sets of transition labels, e.g., nets (a) and (b) have the same transition labels: $\{A, B, C, D, E\}$ while (c) has a smaller set (without B) and (d) has a bigger set (with F). However, models with similar labels can have completely different behaviors (cf. (a) and (b) in Figure 1). Therefore, it is important to consider causal dependencies and the ordering of activities, e.g., to distinguish between parallelism and choice. Another approach could be to consider the state spaces or sets of possible traces of both models. However, in that case the problems are that there may be infinitely many traces/states and that certain paths are more probable.

In this paper, we investigate these problems and propose a completely new approach. *The main idea is to compare two models relative to an event log containing "typical behavior".* This solves several problems when comparing different models. Even models having infinitely many execution sequences can be compared and automatically the relevance of each difference can be taken into account. Moreover, as we will show, we can capture the moment of choice and analyze causalities that may not be explicitly represented in the log.

To give some initial insights in our approach, consider the set of traces listed in Figure 1(f). Each trace represents an execution sequence that may or may not fit in the models at hand. Moreover, frequencies are given, e.g., in the event log trace $ABDE$ occurred 40 times, i.e., there were 40 process instances having this behavior. Figure 1(f) represents some "typical behavior". This may be obtained using simulation of some model or it could be obtained by observing some real-life system/process. All 160 traces fit into the first Petri net (cf. Figure 1(a)), moreover, this Petri net does not allow for any execution sequences not present in the log. In this paper, we will quantify a notion of *fitness*. However, our primary objective is not to compare an event log and a process model, but to compare models in the presence of some event log as shown in Figure 1(f). Compare for example models (a) and (b): in a substantial number of cases (35) D precedes B or C. If we compare (a) and (c) based on the log, we can see that for 55 cases there is a difference regarding the presence of B. We will show that we can *quantify* these differences using the event log. It is important to note that we do not only consider full traces, e.g., if we compare Figure 1(a) with a Petri net where D is missing in the model, there is still some degree of similarity although none of the traces still fits (they all contain D).

This paper extends the results presented in [4] with the application to *genetic mining*. In this paper we show that in the context of genetic mining, i.e., discovering process using genetic algorithms, there is a need to compare models in an approximate manner. Moreover, in this domain the assumption of having example behavior in terms of event logs is very natural.

The remainder is organized as follows. After providing a brief overview of related work, we introduce some preliminaries required to explain our approach. Although we use Petri nets to illustrate our approach, any other process model with some local execution semantics (e.g., EPCs, activity diagrams, BPMN, etc.) could be used. In Section 4, we present two naive approaches (one based on the static structure and one based on a direct comparison of all possible behaviors) and discuss their limitations. Then, in Section 5 we present the core results of this paper. We will show that we can define *precision* and *recall* measures using event logs containing typical behavior. These notions have been implemented in ProM [15]. Finally, we discuss the application of these results to a genetic mining approach and conclude the paper.

2 Overview of Various Equivalence Notations and Related Work

In the literature, many equivalence notions have been defined for process models. Most equivalence notions focus on the dynamics of the model and not on the syntactical structure (e.g., trace equivalence and bisimulation [1, 17, 24]).

This paper uses Petri nets as a theoretical foundation [13, 20, 27]. In [26] an overview is given of equivalence notions in the context of Petri nets. See also [9] for more discussions on equivalence in the context of nets. Most authors translate a Petri net to a transition system to give it semantics. However, there are also authors that emphasize the true-concurrency aspects when giving Petri nets semantics. For example, in [12] the well-known concept of occurrence nets (also named runs) are used to reason about the semantics of Petri nets.

Any model with formal/executable semantics (including Petri nets) can be translated to a (possibly infinite) transition system. If we consider transition systems, many notions of equivalence have been identified. The weakest notion considered is *trace equivalence*: two process models are considered equivalent if the sets of traces they can execute are identical. Trace equivalence has two problems: (1) the set of traces may be infinite and (2) trace equivalence does not capture the moment of choice. The first problem can be addressed in various ways (e.g., looking at finite sets of prefixes or comparing transition systems rather than traces). The second problem requires stronger notions of equivalence. Bisimulation and various kinds of observation equivalence [24] attempt to capture the moment of choice. For example, there may be different processes having identical sets of traces $\{ABC, ABD\}$, e.g., the process where the choice for C or D is made after executing A or the process where the same choice is made only after executing B. Branching bisimilarity [17] is a slightly finer equivalence notion than the well-known observation equivalence [24]. A comparison of branching bisimilarity, observation equivalence, and a few other equivalences on processes with silent behavior can be found in [17]. Branching bisimilarity can be checked in polynomial time (in terms of the size of the transition system) as shown in [18]. Based on these equivalence relations also other relations have been introduced, e.g., the four inheritance relations in [1] are based on branching bisimilarity.

All references mentioned so far, aim at a "true/false" answer. Moreover, they do not take into account that some parts of the process may be more important than others. Few people (e.g., Prakash Panangaden and Jose Desharnais [14]) have been working on probabilistic bisimulation using labeled Markov processes rather than labeled transition systems. See [14] for an excellent overview of this work and also links to the probability theory community working on metrics on spaces of measures. In this paper, we use a different approach. We do not assume that we know any probabilities. Instead we assume that we have some example behavior than can serve as a basis for a comparison of two models. Also related is the work on metric labeled transition systems where the "behavioral difference" between states is a non-negative real number indicating the similarity between those states [10]. This way one can define a behavioral pseudometric to compare transition systems as shown in [10]. Note that this approach very much depends on an explicit notion of states and it is not clear how this can be applied to a practical, mainly activity oriented, setting.

As far as we know, this paper is the first to propose the use of "typical behavior" recorded in event logs as an aid for comparison. This makes the work quite different from the references mentioned in this section. Moreover, we show that this can be used in the context of process mining [3, 6, 5].

3 Preliminaries

This section introduces some of the basic mathematical and Petri-net related concepts used in the remainder.

3.1 Multi-sets, Sequences, and Matrices

Let A be a set. $\mathbb{B}(A) = A \rightarrow \mathbb{N}$ is the set of multi-sets (bags) over A, i.e., $X \in \mathbb{B}(A)$ is a multi-set where for each $a \in A$: $X(a)$ denotes the number of times a is included in the multi-set. The sum of two multi-sets $(X + Y)$, the difference $(X - Y)$, the presence of an element in a multi-set $(x \in X)$, and the notion of subset $(X \leq Y)$ are defined in a straightforward way and they can handle a mixture of sets and multi-sets. The operators are also robust with respect to the domains of the multi-sets, i.e., even if X and Y are defined on different domains, $X + Y$, $X - Y$, and $X \leq Y$ are defined properly by extending the domain where needed. $|X| = \sum_{a \in A} X(a)$ is the size of some multi-set X over A.

For a given set A, A^* is the set of all finite sequences over A. A finite sequence over A of length n is a mapping $\sigma \in \{1, \dots, n\} \to A$. Such a sequence is represented by a string, i.e., $\sigma = \langle a_1, a_2, \dots, a_n \rangle$ where $a_i = \sigma(i)$ for $1 \leq i \leq n$. $hd(\sigma, k) = \langle a_1, a_2, \dots, a_k \rangle$, i.e., the sequence of just the first k elements. Note that $hd(\sigma, 0)$ is the empty sequence.

Every multi-set can be represented as a vector, i.e., $X \in \mathbb{B}(A)$ can be represented as a row vector $(X(a_1), X(a_2), \dots, X(a_n))$ where a_1, a_2, ... ,a_n enumerate the domain of X. $(X(a_1), X(a_2), \dots, X(a_n))^T$ denotes the corresponding column vector (^{T} transposes the vector). Assume X is an $k \times \ell$ matrix, i.e., a matrix with k rows and ℓ columns. A row vector can be seen as $1 \times \ell$ matrix and a column vector can be seen as a $k \times 1$ vector. $X(i, j)$ is the value of the element in the i^{th} row and the j^{th} column. Let X be an $k \times \ell$ matrix and Y an $\ell \times m$ matrix. The product $X \cdot Y$ is the product of X and Y yielding a $k \times m$ matrix, where $X \cdot Y(i, j) = \sum_{1 \leq q \leq \ell} X(i, q)Y(q, j)$. The sum of two matrices having the same dimensions is denoted by $X + Y$.

For any sequence $\sigma \in \{1, \dots, n\} \to A$ over A, the Parikh vector $\overrightarrow{\sigma}$ maps every element a of A onto the number of occurrences of a in σ, i.e., $\overrightarrow{\sigma} \in \mathbb{B}(A)$ where for any $a \in A$: $\overrightarrow{\sigma}(a) = \sum_{1 \leq i \leq n}$ if $\sigma(i) = a$ then 1 else 0.

3.2 Petri nets

This subsection briefly introduces some basic *Petri net* terminology [13, 20, 27] and notations used in the remainder.

Definition 1 (Petri net). *A Petri net is a triple* (P, T, F). P *is a finite set of places,* T *is a finite set of transitions* ($P \cap T = \emptyset$), *and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation).*

Figure 1 shows four Petri nets. Places are represented by circles and transitions are represented by squares.

For any relation/directed graph $G \subseteq A \times A$ we define the preset $\bullet a = \{a_1 \mid (a_1, a) \in G\}$ and postset $a\bullet = \{a_2 \mid (a, a_2) \in G\}$ for any node $a \in A$. We use $\overset{G}{\bullet}\, a$ or $a \overset{G}{\bullet}$ to explicitly indicate the context G if needed. Based on the flow relation F we use this notation as follows. $\bullet t$ denotes the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g., $p\bullet$ is the set of transitions sharing p as an input place. Note that we do not consider multiple arcs from one node to another. In the Petri net shown Figure 1(d): $p5\bullet = \{E, F\}$, $\bullet p5 = \{D\}$, $A\bullet = \{p2, p3\}$, $\bullet A = \{p1\}$, etc.

At any time a place contains zero or more *tokens*, drawn as black dots. The state of the Petri net, often referred to as *marking*, is the distribution of tokens over its places, i.e., $M \in \mathbb{B}(P)$. In each of the four Petri nets shown in Figure 1 only one place is initially marked ($p1$). Note that more places could be marked in the initial state and that places can be marked with multiple tokens.

We use the standard *firing rule*, i.e., a transition t is said to be *enabled* if and only if each input place p of t contains at least one token. An enabled transition may *fire*, and if transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t. For example, in Figure 1(a), A is enabled and firing A will result in the state marking place $p2$ and $p3$. In this state both B, C, and D are enabled. If B fires, C is disabled, but D remains enabled. Similarly, if C fires, B is disabled, but D remains enabled, etc. After firing 4 transitions in Figure 1(a) the resulting state marks $p6$ with one token (independent of the order of B or C). In the following definition, we formalize these notions.

Definition 2 (Firing rule). *Let $N = (P, T, F)$ be a Petri net and $M \in \mathbb{B}(P)$ be a marking.*

- *$enabled(N, M) = \{t \in T \mid M \geq \bullet t\}$ is the set of enabled transitions,*
- *$result(N, M, t) = (M - \bullet t) + t\bullet$ is the state resulting after firing $t \in T$,*
- *$(N, M)[t\rangle(N, M')$ denotes that t is enabled in (N, M) (i.e., $t \in enabled(N, M)$) and that firing t results in marking M' (i.e., $M' = result(N, M, t)$).*

$(N, M)[t\rangle(N, M')$ defines how a Petri net can move from one marking to another by firing a transition. We can extend this notion to firing sequences. Suppose $\sigma = \langle t_1, t_2, \ldots, t_n\rangle$ is a sequence of transitions present in some Petri net N with initial marking M. $(N, M)[\sigma\rangle(N, M')$ means that there is also a sequence of markings $\langle M_0, M_1, \ldots, M_n\rangle$ where $M_0 = M$, $M_n = M'$, and for any $0 \leq i < n$: $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$. Using this notation we define the set of reachable markings $R(N, M)$ as follows: $R(N, M) = \{M' \in \mathbb{B}(P) \mid \exists_\sigma (N, M)[\sigma\rangle(N, M')\}$. Note that $M \in R(N, M)$ because M is reachable via the empty sequence.

Note that $result(N, M, t)$ does not need to yield a multi-set if t is not enabled in marking M because some places may have a negative number of tokens. Although this is not allowed in a Petri net (only enabled transitions can fire), for technical reasons it is sometimes convenient to use markings that may have "negative tokens". This becomes clear when considering the incidence matrix of a Petri net.

Definition 3 (Incidence matrix). *Let $N = (P, T, F)$ be a Petri net and $M \in \mathbb{B}(P)$ be a marking.*

- *$\tilde{N}$ is the incidence matrix of N, i.e., $\tilde{N}$ is a $|P| \times |T|$ matrix with $\tilde{N}(p, t) = 1$ if $(p, t) \notin F$ and $(t, p) \in F$, $\tilde{N}(p, t) = -1$ if $(p, t) \in F$ and $(t, p) \notin F$, and $\tilde{N}(p, t) = 0$ in all other cases,*
- *$result(N, M, \sigma) = M + \tilde{N} \cdot \overrightarrow{\sigma}$ is the state resulting after firing $\sigma \in T^*$,*[1]
- *$enabled(N, M, \sigma) = enabled(N, result(N, M, \sigma))$ is the set of enabled transitions after firing $\sigma \in T^*$.*

The incidence matrix of a Petri net can be used for different types of analysis, e.g., based on $\tilde{N}$ it is possible to efficiently calculate place and transition invariants and to provide minimal (but not sufficient) requirements for the reachability

[1] Note that σ does not need to be enabled, i.e., transitions are forced to fire even if they are not enabled. Also note that we do not explicitly distinguish row and column vectors.

of a marking. It is important to see that *result*(N, M, σ) does not need to yield a valid marking, i.e., there may be a place p such that *result*$(N, M, \sigma)(p) < 0$ indicating a negative number of tokens. If $(N, M)[\sigma\rangle(N, M')$, then *result*$(N, M, \sigma) = M'$. However, the reverse does not need to be the case. *enabled*(N, M, σ) calculates which transitions are enabled *after* firing each transition $\overrightarrow{\sigma}$ times using function *result* and the earlier defined function *enabled* (cf. Definition 2). It may be the case that while executing σ starting from (N, M), transitions were forced to be fired although they were not enabled. As a result, places may get a negative number of tokens. The reason we need such concepts is because we will later compare Petri nets with some observed behavior. In such situations, we need to be able to deal with transitions that were observed even if they were not enabled.

4 Naive Approaches

In this paper we propose to compare two processes on the basis on some event log containing typical behavior. However, before presenting this approach in detail, we first discuss some naive approaches.

4.1 Equivalence of Processes Based on their Structure

When humans compare process models they typically compare the graphical structure, i.e., do the same activities (transitions in Petri net terms) appear in both models and do they have similar connections. Clearly, the graphical structure may be misleading: two models that superficially appear similar may be very different. Nevertheless, the graphical structure is an indicator that may be used to quantify similarity. Let us abstract from the precise split and join behavior (i.e., we do not distinguish between AND/XOR-splits/joins). In other words, we derive a simple graph where each node represents an activity and each arc some kind of connection. For example, the Petri net shown in Figure 1(a) is reduced to a graph with nodes A, B, C, D and E, and arcs (A, B), (A, C), (A, D), (B, E), (C, E) and (D, E). For the other Petri nets models in Figure 1 a similar graph structure can be derived. It is easy to see that each of the four process models has a different graph structure. However, there are many overlapping connections, e.g., all models have arc (A, C). This suggests that from a structural point of view the models are not equivalent but similar. When quantifying the overlap relative to the whole model we can take the perspective of the first model or the second model. This leads to the definition of *precision* and *recall* as specified below.[2]

Definition 4 (Structural Precision and Recall). *Let* $N_1 = (P_1, T_1, F_1)$ *and* $N_2 = (P_2, T_2, F_2)$ *be two Petri nets. Using* $C_1 = \{(t_1, t_2) \in T_1 \times T_1 \mid t_1 \overset{N_1}{\bullet} \cap \overset{N_1}{\bullet}$

[2] These metrics are an adaptation of the precision and recall metrics in [25].

$t_2 \neq \emptyset\}$ *and* $C_2 = \{(t_1, t_2) \in T_2 \times T_2 \mid t_1 \overset{N_2}{\bullet} \cap \overset{N_2}{\bullet} t_2 \neq \emptyset\}$, *we define:*

$$\mathit{precision}^S(N_1, N_2) = \frac{|C_1 \cap C_2|}{|C_2|} \qquad \mathit{recall}^S(N_1, N_2) = \frac{|C_1 \cap C_2|}{|C_1|}$$

$\mathit{precision}^S(N_1, N_2)$ is the fraction of connections in N_2 that also appear in N_1. If this value is 1, the precision is high because all connections in the second model exist in the first model. $\mathit{recall}^S(N_2, N_1)$ is the fraction of connections in N_1 that also appear in N_2. If this value is 1, the recall is high because all connections in the first model appear in the second model. Note that here we think of N_1 as the "original model" and N_2 as some "new model" that we want to compare with the original one.

Let N_a, N_b, N_c, and N_d be the four Petri nets shown in Figure 1.
$\mathit{precision}^S(N_a, N_b) = \frac{|\{(A,B),(A,C),(D,E)\}|}{|\{(A,B),(A,C),(B,D),(C,D),(D,E)\}|} = \frac{3}{5} = 0.6$.
$\mathit{recall}^S(N_a, N_b) = \frac{|\{(A,B),(A,C),(D,E)\}|}{|\{(A,B),(A,C),(A,D),(B,E),(C,E),(D,E)\}|} = \frac{3}{6} = 0.5$.
Note that $\mathit{precision}^S(N_1, N_2) = \mathit{recall}^S(N_2, N_1)$ by definition for any pair of Petri nets N_1 and N_2. Therefore, we only list some precision values: $\mathit{precision}^S(N_a, N_b) = 0.6$, $\mathit{precision}^S(N_a, N_c) = 4/4 = 1.0$, $\mathit{precision}^S(N_a, N_d) = 6/8 = 0.75$, $\mathit{precision}^S(N_b, N_a) = 3/6 = 0.5$, $\mathit{precision}^S(N_b, N_c) = 2/4 = 0.5$, $\mathit{precision}^S(N_b, N_d) = 3/8 = 0.375$, etc. If we consider N_a to be the initial model, then N_c has the best precision of the other three models because all connections in N_c also appear in N_a. Moreover, if we consider N_a to be the initial model, then N_d has the best recall because all connections in N_a also appear in N_d.

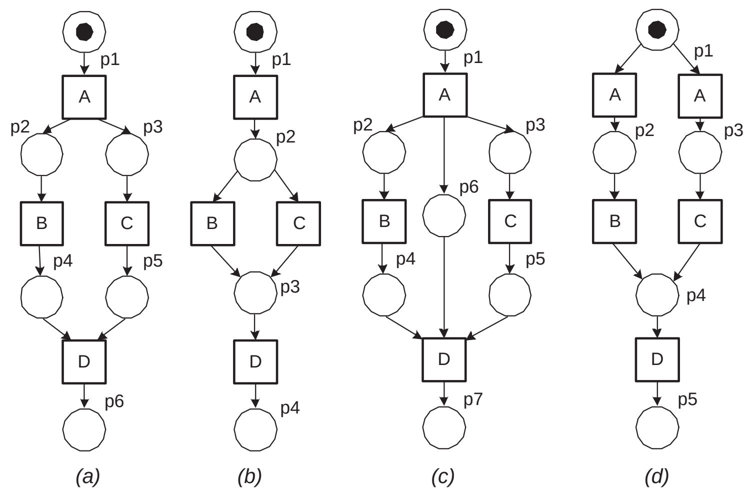


Fig. 2. Although the connection structures of (a) and (b) are similar they are quite different in terms of behavior. Moreover, the connection structure of (a) and (c) differs while the corresponding behaviors are identical.

The precision and recall figures for the four process models in Figure 1 seem reasonable. Unfortunately, models with nearly identical connections may be quite different as is shown in Figure 2. Let N_a, N_b, N_c, and N_d be the four Petri nets shown in Figure 2.[3] Although $\textit{precision}^S(N_a, N_b) = \textit{recall}^S(N_a, N_b) = 1$, N_a and N_b are clearly different. In N_a transitions B and C are executed concurrently while in N_b a choice is made between these two transitions. However, although N_a and N_c are structurally different ($\textit{precision}^S(N_a, N_c) = 4/5 = 0.8$), they have identical behaviors. These examples show that Definition 4 does not provide a completely satisfactory answer when it comes to process equivalence. Nevertheless, $\textit{precision}^S(N_1, N_2)$ and $\textit{recall}^S(N_1, N_1)$ can be used as rough indicators for selecting a similar model, e.g., in a repository of reference models.

4.2 Equivalence of Processes Based on their State Space or Traces

Since process models with a similar structure may have very different behaviors and models with different structures can have similar behaviors, we now focus on *quantifying the equivalence of processes based on their actual behaviors.* We start with a rather naive approach where we define recall and precision based on the *full firing sequences* of two marked Petri nets.

Definition 5 (Naive Behavioral Precision and Recall). *Let $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$ be two Petri nets having initial markings M_1 and M_2 respectively. Moreover, let the corresponding two sets of possible full firing sequences be finite:*
$S_1 = \{\sigma \in T_1^* \mid \exists_{M' \in \mathbb{B}(P_1)} \ (N_1, M_1)[\sigma\rangle(N_1, M') \ \wedge \ \textit{enabled}(N_1, M') = \emptyset\}$ *and*
$S_2 = \{\sigma \in T_2^* \mid \exists_{M' \in \mathbb{B}(P_2)} \ (N_2, M_2)[\sigma\rangle(N_2, M') \ \wedge \ \textit{enabled}(N_2, M') = \emptyset\}$.

$$\textit{precision}^B((N_1, M_1), (N_2, M_2)) = \frac{|S_1 \cap S_2|}{|S_2|}$$

$$\textit{recall}^B((N_1, M_1), (N_2, M_2)) = \frac{|S_1 \cap S_2|}{|S_1|}$$

Clearly, the initial markings of N_1 and N_2 are highly relevant. However, if these are clear from the context, we do not explicitly mention these, i.e., $\textit{precision}^B(N_1, N_2) = \textit{precision}^B((N_1, M_1), (N_2, M_2))$ and $\textit{recall}^B(N_1, N_2) = \textit{recall}^B((N_1, M_1), (N_2, M_2))$.

Let N_a, N_b, N_c, and N_d be the four Petri nets shown in Figure 2 and S_a, S_b, S_c, and S_d their corresponding full firing sequences. $S_a = \{\langle A, B, C, D\rangle, \langle A, C, B, D\rangle\}$, $S_b = \{\langle A, B, D\rangle, \langle A, C, D\rangle\}$, $S_c = S_a$, and $S_d = S_b$. Hence, $\textit{precision}^B(N_a, N_b) = 0$ and $\textit{recall}^B(N_a, N_b) = 0$, i.e., the models are considered to be completely different because there are no identical full firing sequences possible in both models.

[3] Note that strictly speaking N_d does not correspond to a Petri net as defined in Definition 1, because there are two transitions A. However, it is easy to extend Definition 1 to so-called labeled Petri nets where different transitions can have the same label.

However, $\mathit{precision}^B(N_a, N_c) = 1$ and $\mathit{recall}^B(N_a, N_c) = 1$ and $\mathit{precision}^B(N_b, N_d) = 1$ and $\mathit{recall}^B(N_b, N_d) = 1$.

We can also consider the four process models in Figure 1. The fourth model (N_d) has an infinite set of full firing sequences. Therefore, we focus on the first three models: N_a, N_b, and N_c. Let us first compare N_a and N_b: $\mathit{precision}^B(N_a, N_b) = 2/2 = 1$ and $\mathit{recall}^B(N_a, N_b) = 2/4 = 0.5$, i.e., all full firing sequences in N_b are possible in N_a but not the other way around. Although N_c differs from N_b, the precision and recall values are identical when comparing with N_a, i.e., $\mathit{precision}^B(N_a, N_c) = 1$ and $\mathit{recall}^B(N_a, N_c) = 0.5$.

These examples show that Definition 5 provides another useful quantification of equivalence quite different from Definition 4. However, also this quantification has a number of problems:

1. The set of full firing sequences needs to be *finite*. This does not need to be the case as is illustrated by the Petri net shown in Figure 1(d). For such models, the metric becomes useless.
2. The models need to be *terminating*, i.e., it should be possible to end in a dead marking representing the completion of the process. Note that models may have unintentional livelocks or are designed to be non-terminating. For such models, we cannot apply Definition 5 in a meaningful way. It also does not make sense to look at all possible firing sequences (i.e., also firing sequences that are not *full* firing sequences), because this would include the prefixes of both terminating and non-terminating sequences. As a result, new problems are introduced, e.g., more emphasis on the behavior typically contained in prefixes and possibly infinite sets.
3. Definition 5 does not take into account differences between important paths or parts versus unimportant paths or parts of the model. For example, certain full firing sequences may have a very *low probability* in comparison to other sequences that occur *more frequent*. There may be parts of the process model that are rarely activated (earlier named "process veins") while other parts are executed for all process instances (earlier named "process arteries"). Clearly this should be taken into account.
4. Fourth, Definition 5 appears to be too *rigid*, i.e., one difference in a full firing sequence invalidates the entire sequence. In Figure 2 $\mathit{precision}^B(N_a, N_b) = 0$ and $\mathit{recall}^B(N_a, N_b) = 0$ although both models always start with A and end with D.
5. The *moment of choice* is not taken into account in Definition 5, i.e., essentially trace equivalence is used as a criterion. Many authors [1, 17, 24] have emphasized the importance of preserving the moment of choice by defining notions such as observation equivalence, bisimilarity, branching/weak bisimilarity, etc. To illustrate the importance of preserving the moment of choice, consider N_b and N_d depicted in Figure 2. Although $\mathit{precision}^B(N_b, N_d) = 1$ and $\mathit{recall}^B(N_b, N_d) = 1$, most environments will be able to distinguish both processes. In N_b in Figure 2(b) there is no state where only B or just C is enabled. However, such states exist in N_d in Figure 2(d), e.g., there can be a token in $p2$ enabling only B. Suppose that B and C correspond to the

receipt of different messages sendt by some environment. In this case, N_d potentially deadlocks, e.g., a message for B cannot be handled because the system is waiting for C (i.e., $p3$ is marked). Such a deadlock is not possible in N_b.

The problems listed above show that similarity metrics based on criteria directly comparing all possible behaviors in terms of traces are of little use from a practical point of view. An alternative approach is to compare the state spaces rather than the sets of traces. For example, trying to establish a bisimulation relation where states are related in such a way that any move of one process model can be followed by the other one and vice versa [1, 17, 24]. However, this would only solve some of the problems listed above. Moreover, the notion of state often only exists implicitly and it is very difficult to extend more refined equivalence notions to include probabilities (cf. [10, 14]). Therefore, we propose another approach as presented in the next section.

5 Equivalence of Processes in the Context of Observed Behavior

To overcome the problems highlighted so far, we propose an approach that uses *exemplary behavior* to compare two models. This exemplary behavior can be obtained on the basis of real process executions (in case the process already exists), user-defined scenarios, or by simply simulating one of the two models (or both). We assume this exemplary behavior to be recorded in an *event log.*

Definition 6 (Event log). *An event log L is a multi-set of sequences on some set of T, i.e., $L \in \mathbb{B}(T^*)$.*

An event log can be considered as a multi-set of full firing sequences (cf. Definition 5). However, now these sequences may exist independent of some model and the same sequence may occur multiple times.

Before comparing two process models using an event log, we first define the notion of *fitness.* This notion is inspired by earlier work on genetic mining and conformance checking [23, 29].

Definition 7 (Fitness). *Let (N, M) be a marked Petri net and let $L \in \mathbb{B}(T^*)$ be a multi-set over T.*[4]

$$\mathit{fitness}((N,M),L) =$$
$$\Big(\sum_{\sigma \in L} \frac{L(\sigma)}{|\sigma|} \;\; |\{i \in \{0, |\sigma|-1\} \mid \sigma(i+1) \in \mathit{enabled}(N,M,\mathit{hd}(\sigma,i))\}| \;\; \Big)/|L|$$

[4] Note that not all events in the log need to correspond to actual transitions. These events are simply ignored, i.e., we assume $\mathit{enabled}(N, M, \sigma)$ to be defined properly even if not all transitions in σ actually appear in N.

fitness$((N, M), L)$ yields a number between 0 and 1. Note that per sequence $\sigma \in L$ we calculate the number of times that a transition that was supposed to fire according to σ was actually enabled. This is divided by $|\sigma|$ to yield a number between 0 and 1 per sequence. This number shows the "fit" of σ. This is repeated for all $\sigma \in L$. Since the same sequence may appear multiple times in L (i.e., $L(\sigma) > 1$), we multiply the result for σ with $L(\sigma)$ and divide by $|L|$. Definition 7 assumes that $|L| > 0$ and $|\sigma| > 0$. This is not a fundamental restriction, if such strange cases occur (empty event log or an empty sequence), then we can simply assume that $0/0 = 0$.

As an example, consider the event log L shown in Figure 1(f) containing 160 traces. Clearly, *fitness*$(N_a, L) = 1$ because all sequences in L can be reproduced by N_a.[5] Moreover, *fitness*$(N_b, L) = (40 + 85 + (15 * 3/4) + (20 * 3/4))/160 = 0.945$, *fitness*$(N_c, L) = ((40 * 1/2) + 85 + (15 * 1/2) + 20)/160 = 0.828$, and *fitness*$(N_d, L) = 1$. These examples show that Definition 7 matches our intuitive understanding of fitness. *It is important to note that transitions are "forced" to fire even if they are not enabled*, cf. Definition 3. Moreover, a particular sequence can be "partly fitting", e.g., if we parse sequence $\langle A, B, D, E\rangle$ using N_c in Figure 1(c), half of the sequence fits. When forcing the execution of $\langle A, B, D, E\rangle$ using N_c, A is initially enabled. However, B is not enabled and does not even exist in the model. Nevertheless, in the resulting state D is still enabled. However, after firing D, the last event in the sequence (E) is not enabled. Hence, only two of the four events in $\langle A, B, D, E\rangle$ are actually enabled, resulting in a fitness of 0.5. Note that it is better to look at individual events rather than considering whole sequences like in Definition 5. Using Definition 7, *fitness*$(N_c, L) = 0.828$. However, if we would focus on completely fitting sequences, *fitness*$(N_c, L) = (0 + 85 + 0 + 20)/160 = 0.656$, i.e., considerably lower because partly fitting are ignored.

Inspired by the definition of fitness, we would like to compare *two* models on the basis of a log. A straightforward extension of Definition 7 to two models is to compare the overlap in fitting or partially fitting sequences. However, in this case one only considers the actual behavior contained in the log. Therefore, we go one step further and look at the *enabled transitions* in both models and compare these, i.e., we do not just check whether an event in some sequence is possible, but *also take into account all enabled transitions at any point in the sequence.* This idea results in the following definition of precision and recall.

Definition 8 (Behavioral Precision and Recall). *Let* (N_1, M_1) *and* (N_2, M_2) *be marked Petri nets and let* $L \in \mathbb{B}(T^*)$ *be a multi-set over* T.[6]

precision$((N_1, M_1), (N_2, M_2), L) =$

[5] Note that again we omit the initial marking if it is clear from the context, i.e., *fitness*$(N_a, L) =$ *fitness*$((N_a, [p1]), L)$.

[6] Note that the two denominators $|enabled(N_2, M_2, hd(\sigma, i))|$ and $|enabled(N_1, M_1, hd(\sigma, i))|$ may evaluate to zero. In these case, the numerator is also zero. Again, we assume in such cases that $0/0 = 0$.

$$(\sum_{\sigma\in L}\frac{L(\sigma)}{|\sigma|}(\sum_{i=0}^{|\sigma|-1}\frac{|\mathit{enabled}(N_1,M_1,\mathit{hd}(\sigma,i))\cap \mathit{enabled}(N_2,M_2,\mathit{hd}(\sigma,i))|}{|\mathit{enabled}(N_2,M_2,\mathit{hd}(\sigma,i))|}\ )\ )/|L|$$

$\mathit{recall}((N_1,M_1),(N_2,M_2),L) =$

$$(\sum_{\sigma\in L}\frac{L(\sigma)}{|\sigma|}(\sum_{i=0}^{|\sigma|-1}\frac{|\mathit{enabled}(N_1,M_1,\mathit{hd}(\sigma,i))\cap \mathit{enabled}(N_2,M_2,\mathit{hd}(\sigma,i))|}{|\mathit{enabled}(N_1,M_1,\mathit{hd}(\sigma,i))|}\ )\ )/|L|$$

To explain the concept consider a log $L = \{(\langle A,B,C,D\rangle,2),(\langle A,C,B,D\rangle,1)\}$ and the first three Petri nets shown in Figure 2. $\mathit{precision}(N_a,N_b,L) = ((2/4*(1/1+2/2+0/1+1/1)) + (1/4*(1/1+2/2+0/1+1/1)))/3 = 0.75$ and $\mathit{recall}(N_a,N_b,L) = ((2/4*(1/1+2/2+0/1+1/1)) + (1/4*(1/1+2/2+0/1+1/1)))/3 = 0.75$. $\mathit{precision}(N_a,N_c,L) = \mathit{recall}(N_a,N_c,L) = 1$.

We can also consider the four process models in Figure 1 with respect to the logs shown in Figure 1(f). $\mathit{precision}(N_a,N_b,L) = ((40/4*(1/1+2/2+1/1+1/1)) + (85/4*(1/1+2/2+1/1+1/1)) + (15/4*(1/1+2/2+2/3+1/1)) + (20/4*(1/1+2/2+2/3+1/1)))/160 = 0.98$ and $\mathit{recall}(N_a,N_b,L) = ((40/4*(1/1+2/3+1/1+1/1)) + (85/4*(1/1+2/3+1/1+1/1)) + (15/4*(1/1+2/3+2/2+1/1)) + (20/4*(1/1+2/3+2/2+1/1)))/160 = 0.92$. Note that both numbers would be lower if the sequences starting with $\langle A,D,\ldots\rangle$ would be more frequent. Let us now compare N_a and N_d in Figure 1 using L. $\mathit{precision}(N_a,N_d,L) = ((40/4*(1/1+3/3+1/1+1/2))+(85/4*(1/1+3/3+1/1+1/2))+(15/4*(1/1+3/3+2/3+1/2))+(20/4*(1/1+3/3+2/3+1/2)))/160 = 0.75$ and $\mathit{recall}(N_a,N_d,L) = ((40/4*(1/1+3/3+1/1+1/1))+(85/4*(1/1+3/3+1/1+1/1)) + (15/4*(1/1+3/3+2/2+1/1)) + (20/4*(1/1+3/3+2/2+1/1)))/160 = 1$. Note that N_d allows for behavior not present in log L (i.e., executing F). Nevertheless, as we can see from $\mathit{precision}(N_a,N_d,L) = 0.75$, the enabling of F is taken into account. It is also easy to see that Definition 8 takes into account the moment of choice, i.e., the enabling of set of transitions is the basis of comparison rather than the resulting sequences. Hence, we can distinguish N_b and N_d in Figure 2.[7]

In Section 4.2 we listed five problems related to the use of Definition 5. It is easy to see that Definition 8 addresses each of these problems:

1. Even models with an infinite set of firing sequences can be compared using a finite, but representative, set of traces.
2. Models do not need to be terminating.
3. Differences between frequent and infrequent sequences can be taken into account by selecting a representative log.
4. Partial fits are taken into account, i.e., small local differences do not result in a complete "misfit".
5. The moment of choice is taken into account because the focus is on enabling.

[7] Note that N_d contains duplicate labels, i.e., two transitions with label A. However, it is possible to extend Definition 8 and the resulting approach for such models.

Given the attractive properties of the precision and recall metrics defined in Definition 8, we have implemented these metrics in the ProM framework [15].[8] Here it has been applied to a variety of process models as will be discussed in Section 6.

One the of critical success factors is the availability of some log L that can serve as a basis for comparison. We propose to use *existing event logs* or to generate *artificial logs using simulation.*

Existing logs can be extracted from information systems but can also be obtained by manually describing some typical scenarios. It is important to realize that today's information systems are logging a wide variety of events. For example, any user action is logged in ERP systems like SAP R/3, workflow management systems like Staffware, and case handling systems like FLOWer. Classical information systems have some centralized database for logging such events (called transaction log or audit trail). Modern service-oriented architectures record the interactions between web services (e.g., in the form of SOAP messages). Moreover, today's organizations are forced to log events by national or international regulations (cf. the Sarbanes-Oxley (SOX) Act [30] that is forcing organizations to audit their processes).

An example application scenario where existing event logs are used is the comparison of an existing process and a set of possible redesigns. For each of the redesigns, we can measure the precision and recall taking an event log of the existing information system as a starting point. First of all, the existing process can be compared with this event log using the fitness notion presented in this section. This gives an indication of the quality of the initial model. Then, if the quality is acceptable, each of the redesigns can be compared with the existing process using this log.

Another approach would be to use simulation. This simulation could be based on both models or just the initial model. Note that the generated logs do not need to be complete, because Definition 8 also takes the enabling into account. It is more important that the probabilities are taken into account, because differences in the "process veins" are of less importance than differences in the "process arteries".

6 Application to Genetic Mining

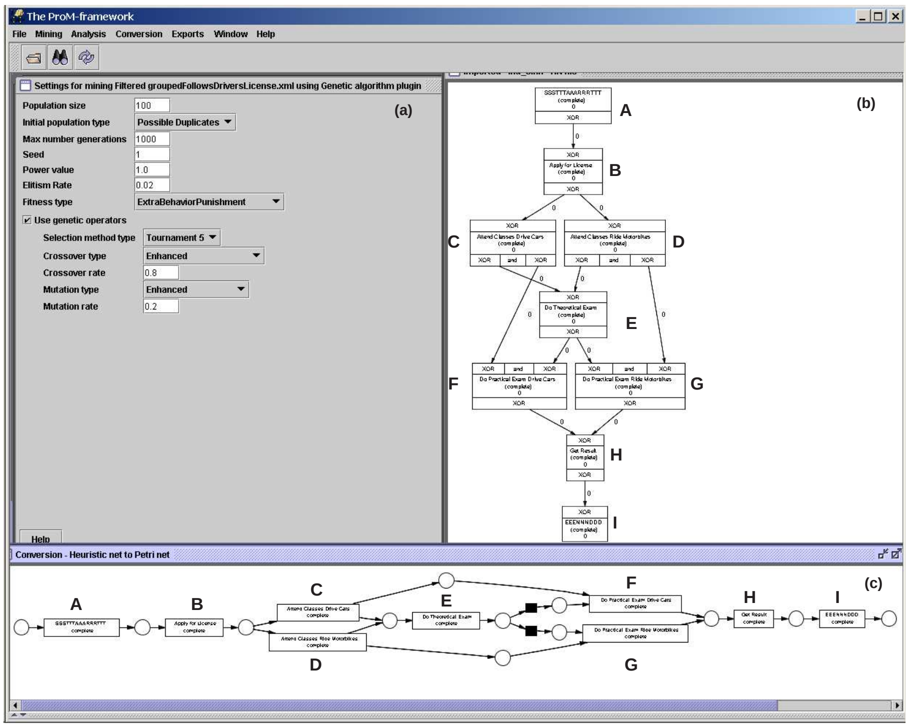
Process mining aims at extracting information from event logs to capture the business process as it is being executed. Process mining is particularly useful in situations where events are recorded but there is no system enforcing people to work in a particular way. Consider for example a hospital where the diagnosis and treatment activities are recorded in the hospital information system, but where health-care professionals determine the "careflow". A variety of process mining algorithms have developed [5–7, 11, 19], including our approach based on genetic process mining [3, 23].

[8] ProM and the analysis plug-in implementing the precision and recall metrics can be downloaded from www.processmining.org.

The goal of process mining is to extract information about processes from event logs [5], e.g., a log like the one shown in Figure 1(f). The result is a model, e.g., a Petri net. Using simple approaches such as the one presented in [6] it is possible to automatically discover the Petri net shown in Figure 1(a) based on the log shown in Figure 1(f). Unfortunately, existing approaches for mining the process perspective have problems dealing with issues such as duplicate activities, hidden activities, non-free-choice constructs, noise, and incompleteness. The problem with *duplicate activities* occurs when the same activity can occur at multiple places in the process. This is a problem because it is no longer clear to which activity some event refers. The problem with *hidden activities* is that essential routing decisions are not logged but impact the routing of cases. *Non-free-choice* constructs are problematic because it is not possible to separate choice from synchronization. We consider two sources of *noise*: (1) incorrectly logged events (i.e., the log does not reflect reality) or (2) exceptions (i.e., sequences of events corresponding to "abnormal behavior"). Clearly noise is difficult to handle. The problem of *incompleteness* is that for many processes it is not realistic to assume that all possible behavior is contained in the log. The goal of genetic process mining [3, 23] is to overcome these problems. A genetic algorithm starts with an initial population of individuals (in this case process models). Populations evolve by selecting the fittest individuals and generating new individuals using genetic operators such as *crossover* (combining parts of two of more individuals) and *mutation* (random modification of an individual). Using proper notions of fitness, crossover, and mutation, populations evolve and improve until the "best fitting" model is discovered. The "best fitting" discovered model that the genetic algorithm targets at is *complete* (can parse all the traces in the log) and *precise* (does not allow for the parsing of too much extra behavior that cannot be derived from the behavior observed in the log). Figure 3 shows a screenshot of the *Genetic algorithm* plug-in in the ProM framework. The ProM framework can be downloaded from www.processmining.org and supports the development of plug-ins to mine event logs.

One of the problems of doing research in this area is that it is difficult to judge the result. For *testing* our approach we do not only take real-life logs (e.g., from SAP, Staffware or FLOWer) but also generate logs from known models. In the later case, *we can compare the "original model"* (i.e., the initial model used to generate logs from) *with the "discovered model"* (i.e., the one obtained using process mining). Experience shows that typically the initial model and the discovered model are not identical: the structure is different and/or behaviorally there are also differences. In the genetic algorithm setting, these differences can occur because, for instance, there is more than one complete and precise model that can be discovered for a given log, or simply because the genetic algorithm did not find a very precise model. Hence, to measure the quality of mining result we need to consider the issues raised in this paper. Moreover, in this setting there are event logs that can serve as a basis for comparison.

Figures 3 and 4 show some results obtained by applying the genetic algorithm to the log $L = \{(\langle A, B, C, E, F, H, I \rangle, 141), (\langle A, B, D, E, G, H, I \rangle, 159)\}$. Figure 3


The ProM-framework
File Mining Analysis Conversion Exports Window Help
Help
Conversion - Heuristic net to Petri net
(a)
(b)
(c)
A
B
C
D
E
F
G
H
I

Settings for mining Filtered groupedFollowsDriversLicense.xml using Genetic algorithm plugin

Population size	100
Initial population type	Possible Duplicates
Max number generations	1000
Seed	1
Power value	1.0
Elitism Rate	0.02
Fitness type	ExtraBehaviorPunishment

☑ Use genetic operators

Selection method type	Tournament 5
Crossover type	Enhanced
Crossover rate	0.8
Mutation type	Enhanced
Mutation rate	0.2

Fig. 3. Screenshot of the ProM framework showing three aspects of the *Genetic algorithm* plug-in: (a) the configuration window allows for the setting of parameters, (b) the discovered model in terms of the internal representation, and (c) the corresponding

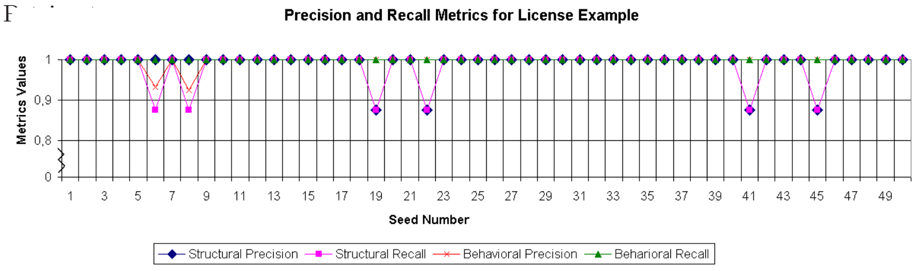
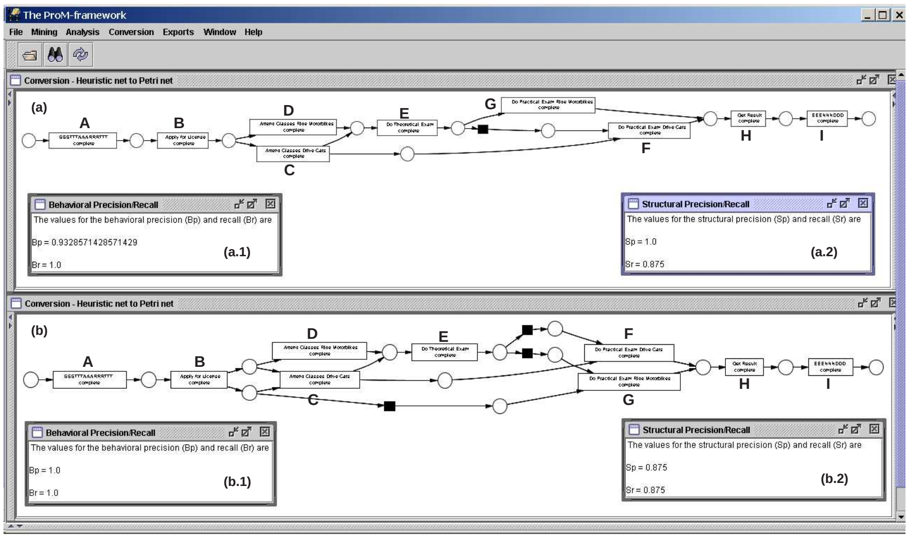


Fig. 4. Precision and recall values for the best fitting individual mined by the genetic algorithm, for 50 different seed values.

shows the result of one run of the genetic algorithm in ProM. Figure 4 shows the precision and recall values over 50 runs. The results show that the genetic algorithm found a model with the same structure and behavior as the original model for most of the runs. The Petri net representation of the mined model for these runs looks like the one in Figure 3(c). Note that this Petri net shows a non-free-choice construct involving the tasks C, D, E, F and G. In other words, the choice of whether executing F or G is not done after executing E, but when executing C or D. Note that the behavior in the log tell us that the task F is only executed when the task C has been executed. A similar observation holds for tasks G and B. Figure 4 shows that the genetic algorithm did pretty well for 88% (44/50) of the runs. However, for six runs (run 6, 8, 19, 22, 41 and 45), the genetic algorithm mined a different model. In runs 6 and 8, the genetic algorithm found a model that is a substructure of the original model ($precision^S = 1$ and $recall^S = 0.875$) and allows for more behavior than the original model ($precision = 0.933$ and $recall = 1$). The mined model for both runs was exactly the same. The Petri net representation for this mined model is shown in Figure 5(a). Figure 5(b) shows the mined model for runs 19, 22, 41 and 45. Note that, as indicated by the metrics, this mined model has the same behavior as the original one ($precision = 1$ and $recall = 1$) and a different structure ($precision^S = 0.875$ and $recall^S = 0.875$). The use of the structural and precision metrics allows us to measure how well the genetic algorithm is doing when mining process models. The metrics make it possible to analyze the similarity of the original and mined models in terms of possible behavior and thus *quantify* their differences.


The ProM-framework
File Mining Analysis Conversion Exports Window Help
Conversion - Heuristic net to Petri net
(a)
A
SSSTTTAAARRRTTT complete
B
Apply for License complete
D
Attend Classes Ride Motorbikes complete
C
Attend Classes Drive Cars complete
E
Do Theoretical Exam complete
G
Do Practical Exam Ride Motorbikes complete
F
Do Practical Exam Drive Cars complete
H
Get Result complete
I
EEENNNDDD complete
Behavioral Precision/Recall
(a.1)
Structural Precision/Recall
(a.2)
Conversion - Heuristic net to Petri net
(b)
A
B
D
C
E
F
G
H
I
Behavioral Precision/Recall
(b.1)
Structural Precision/Recall
(b.2)

(a.1) The values for the behavioral precision (Bp) and recall (Br) are

Bp = 0.9328571428571429

Br = 1.0

(a.2) The values for the structural precision (Sp) and recall (Sr) are

Sp = 1.0

Sr = 0.875

(b.1) The values for the behavioral precision (Bp) and recall (Br) are

Bp = 1.0

Br = 1.0

(b.2) The values for the structural precision (Sp) and recall (Sr) are

Sp = 0.875

Sr = 0.875

Fig. 5. Screenshot of the structural and behavioral metrics in the ProM framework for two different mined models. Note that the original model is the Petri net shown in Figure 3(c).

7 Conclusion

This paper presented a novel approach to compare process models. Existing approaches typically do not quantify equivalence, i.e., models are equivalent or not. However, for many practical applications such an approach is not very useful, because in most real-life settings we want to distinguish between marginally different processes and completely different processes. We have proposed and implemented notions of *fitness*, *precision*, and *recall* in the context of the ProM framework. The key differentiator is that these notions take an event log with typical execution sequences as a starting point. This allows us to overcome many of the problems associated with approaches directly comparing processes at the model level. Although our approach is based on Petri nets, it can be applied to other models with executable semantics, e.g., formalizations of EPCs, BPMN, or UML activity diagrams.

We have applied the approach in the context of process mining. However, the notions of precision and recall can be applied in a wide variety of situations, e.g., to measure the difference between an organization specific process model and a reference model, to select a web service that fits best based on some description (e.g., PIPs or abstract BPEL), or to compare an existing process model with some redesign. In our future work, we would like to explore more of these applications, e.g., comparing clinical guidelines.

References

1. W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.
2. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems.* MIT press, Cambridge, MA, 2004.
3. W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, Berlin, 2005.
4. W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Process Equivalence: Comparing Two Process Models Based on Observed Behavior. In S. Dustdar, J. Fiadeiro and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2006 (to appear).
5. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
6. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
7. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
8. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.

9. E. Best and M.W. Shields. Some equivalence results for free choice nets and simple nets, and on the periodicity of live free choice nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Proceedings of CAAP '83*, volume 159 of *Lecture Notes in Computer Science*, pages 141–154. Springer-Verlag, Berlin, 1987.
10. F. van Breugel. A Behavioural Pseudometric for Metric Labelled Transition Systems. In *16th International Conference on Concurrency Theory (CONCUR 2005)*, volume 3653 of *Lecture Notes in Computer Science*, pages 141–155. Springer-Verlag, Berlin, 2005.
11. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
12. J. Desel. Validation of Process Models by Construction of Process Nets. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 110–128. Springer-Verlag, Berlin, 2000.
13. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, Cambridge, UK, 1995.
14. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labelled Markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.
15. B. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
16. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology.* Wiley & Sons, 2005.
17. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
18. J.F. Groote and F.W. Vaandrager. An Efficient Algorithm for Branching and Stuttering Equivalence. In M.S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer-Verlag, Berlin, 1990.
19. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
20. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1997.
21. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Choreography Description Language, Version 1.0. W3C Working Draft 17-12-04, 2004.
22. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation.* Addison-Wesley, Reading MA, 1998.
23. A.K.A. de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: A Basic Approach and its Challenges. In C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 203–215. Springer-Verlag, Berlin, 2006.

24. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1980.
25. S.S. Pinter and M. Golani. Discovering Workflow Models from Activities Lifespans. *Computers in Industry*, 53(3):283–296, 2004.
26. L. Pomello, G. Rozenberg, and C. Simone. A Survey of Equivalence Notions of Net Based Systems. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 420–472. Springer-Verlag, Berlin, 1992.
27. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
28. RosettaNet. RosettaNet Partner Interface Processes (PIPs). www.rosettanet.org, 2006.
29. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.
30. P. Sarbanes and G. Oxley et. al. Sarbanes-Oxley Act of 2002, 2002.
31. A. Wombacher and B. Mahleko. Finding Trading Partners to Establish Ad-hoc Business Processes. In *On the Move to Meaningful Internet Systems, 2002 - Proceedings of the DOA/CoopIS/ODBASE Confederated International Conferences*, pages 339–355, Irvine CA, USA, October 2002. Springer Verlag.