Log-Based Simplification of Process Models

Javier De San Pedro, Josep Carmona, and Jordi Cortadella

Universitat Politècnica de Catalunya, Barcelona (Spain)

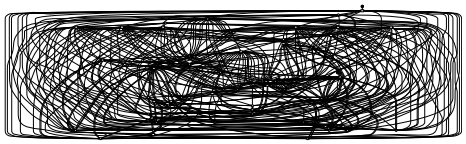
Abstract. The visualization of models is essential for user-friendly human-machine interactions during Process Mining. A simple graphical representation contributes to give intuitive information about the behavior of a system. However, complex systems cannot always be represented with succinct models that can be easily visualized. Quality-preserving model simplifications can be of paramount importance to alleviate the complexity of finding useful and attractive visualizations.
This paper presents a collection of log-based techniques to simplify process models. The techniques trade off visual-friendly properties with quality metrics related to logs, such as fitness and precision, to avoid degrading the resulting model. The algorithms, either cast as optimization problems or heuristically guided, find simplified versions of the initial process model, and can be applied in the final stage of the process mining life-cycle, between the discovery of a process model and the deployment to the final user. A tool has been developed and tested on large logs, producing simplified process models that are one order of magnitude smaller while keeping fitness and precision under reasonable margins.

1 Introduction

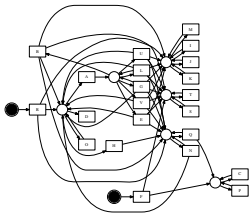
The understandability of a process model can be seriously hampered by a poor visualization. Many factors may contribute to this, being complexity a crucial one: models that are unnecessarily complex (incorporating redundant components, or components with limited importance) are often not useful for understanding the process behind. On the other hand, process models are expected to satisfy certain quality metrics when representing an event log: *fitness*, *precision*, *simplicity* and *generalization* [1]. In this paper we present techniques to simplify a process model while retaining the aforementioned quality metrics under reasonable margins. We focus on the simplification of Petri nets, a general formalism onto which several other process models can be essentially mapped.

Given a complex process model, one can simply remove arcs and nodes until a nice graphical object is obtained. However, this naive technique has two main drawbacks. First, the capability of the simplified model to replay the process executions may be considerably degraded, thus deriving a highly unfitting model. Second, the model components, arcs and places in a Petri net, are not equally important when replaying process executions, and therefore one may be interested in keeping those components that provide more insight into the real boundaries on what is allowed by the process (i.e., its precision).

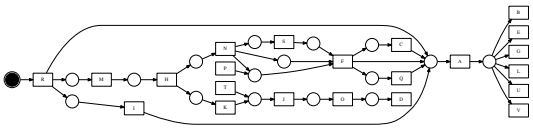
Given a Petri net and an event log, this paper first ranks the importance of places and arcs using a simple simulation of the log by the Petri net, and then



(a) Initial process model.



(b) Simplified fitting proc. model.



(c) Simplified series-parallel process model.

Fig. 1. Log-based simplification of an spaghetti-like process model.

simplifies the model by retaining those arcs and places that are important in restricting the behavior allowed by the model. Several alternatives are presented, which render the preservation of fitness as a user decision, or extract certain Petri net subclasses (State Machines, Free-Choice) or structural subclasses (Series-Parallel graphs). The goal of these techniques is similar to that of [2]. However the techniques presented in this paper require a lower computational cost and exhibit better scalability when managing large problems, while producing results of competitive quality, as shown by the experimental results.

1.1 Motivating Example

We will illustrate one of the techniques presented in this paper with the help of an example. We have used the general-purpose tool `dot` [3] to render all the examples. Figure 1a reports a process model that has been discovered by the ILP miner from a real-life log, a well-known method for process discovery [4]. This miner guarantees perfect fitness (i.e., the model is able to reproduce all the traces in the log), but its precision value is low (31.5%) which indicates that the model may generate many traces not observed in the log.

Clearly, this model does not give any insight about the executions of the process behind. Hence, although it is a model having perfect fitness, some of the other quality metrics (precision, simplicity) are not satisfactory. Applying the simplification techniques of this paper, a process model can be transformed with the objective of improving its understandability. The process models at the bottom of Fig. 1 are the result of applying two of the techniques proposed in this paper. In Figure 1b the model is simplified while preserving as much as possible the quality metrics of the original model. The model has 6 times fewer places and arcs, making it much easier to understand. The resulting fitness is still perfect, but the precision has been reduced to 22.5%.

In Figure 1c we reduce the model to a *series-parallel graph*, further improving its simplicity and understandability. The fitness has been reduced to 64.1%, but on the other hand its precision has improved considerably (now 48.7%).

The paper is organized as follows. Section 2 introduces the required background of the paper. Section 3 gives an overview of the proposed simplification algorithms. In Section 4, a log-based technique to estimate the importance of arcs and places is described, which is used by some of the simplification algorithms, detailed in Section 5. Section 6 describes the remaining, non log-based simplification techniques. All techniques are evaluated in Section 7. Finally, related work and conclusions are discussed in Sections 8 and 9, respectively.

2 Preliminaries

2.1 Process Models

Process models are formalisms to represent the behavior of a process. Among the different formalisms, Petri nets are perhaps the most popular, due to its well-defined semantics. In this paper we focus on visualization of Petri nets, although the work may be adapted to other formalisms like BPMN, EPC or similar.

A Petri Net [5] is a 4-tuple $N = \langle P, T, \mathcal{F}, m_0 \rangle$, where P is the set of places, T is the set of transitions, $\mathcal{F} : (P \times T) \cup (T \times P) \to \{0, 1\}$ is the flow relation, and m_0 is the initial marking. A marking is an assignment of a non-negative integer to each place. If k is assigned to place p by marking m (denoted $m(p) = k$), we say that p is marked with k tokens. Given a node $x \in P \cup T$, its pre-set and post-set are denoted by ${}^\bullet x$ and $x^\bullet$ respectively.

A transition t is *enabled* in a marking m when all places in ${}^\bullet t$ are marked. When t is enabled, it can *fire* by removing a token from each place in ${}^\bullet t$ and putting a token to each place in $t^\bullet$. A marking m' is *reachable* from m if there is a sequence of firings $t_1 t_2 \dots t_n$ that transforms m into m', denoted by $m[t_1 t_2 \dots t_n \rangle m'$. A sequence $t_1 t_2 \dots t_n$ is *feasible* if it is firable from m_0. A Petri net N is a: *Marked graph* if $\forall p \in P : |{}^\bullet p| = |p^\bullet| = 1$, *State machine* if $\forall t \in T : |{}^\bullet t| = |t^\bullet| = 1$ and *Free-Choice* if $\forall p_1, p_2 \in P : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow |p_1^\bullet| = |p_2^\bullet| = 1$.

2.2 Process Mining

A *trace* is a word $\sigma \in T^*$ that represents a finite sequence of events. An *event log* $L \in \mathcal{B}(T^*)$ is a multiset of traces[1]. Event logs are the starting point to apply process mining techniques, guided towards the discovery, analysis or extension of process models. *Process discovery* is one of the most important disciplines in process mining, concerned with learning a process model (e.g., a Petri net) from an event log. Although a novel discipline, there are several discovery techniques that have appeared in the last decade, most of them summarized in [1].

The second family of techniques in process mining is *conformance checking*, i.e., comparing observed and modeled behavior. There are four quality dimensions for comparing model and log: (1) *replay fitness*, (2) *simplicity*, (3) *precision*, and (4) *generalization* [1]. A model has a perfect replay fitness if all traces in the log can be replayed by the model from beginning to end. The *simplest* model that can explain the behavior seen in the log is the best model, a principle known

[1] $\mathcal{B}(A)$ denotes the set of all multisets over A.

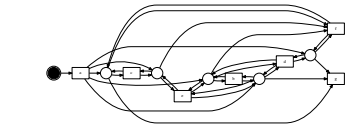
as Occam's Razor. Fitness and simplicity alone are not sufficient to judge the quality of a discovered process model. For example, it is very easy to construct an extremely simple Petri net ("flower model") that is able to replay all traces in an event log (but also any other event log referring to the same set of activities). Similarly, it is undesirable to have a model that only allows for the exact behavior seen in the event log. A model is *precise* if it does not contain "too much" behavior that has not been observed in the log. A model that is not precise is "underfitting"[6]. In contrast to precision, a model should generalize and not restrict behavior to just the examples seen in the log.

In this paper, we consider simplifications that may preserve replay fitness which we will simply refer to as *fitness*. Metrics for fitness have been defined as indicators of how every trace in the log fits a model [7]. Likewise, metrics for precision exist in the literature [6].

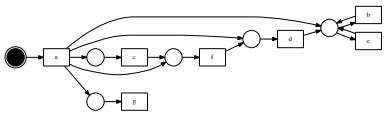
Definition 1 (Fitting Trace and Log). *A trace $\sigma \in T^*$ fits a Petri net N if σ is a feasible sequence in N. An event log L fits N if for all $\sigma \in L$, σ fits N.*

3 Overview of Proposed Simplification Techniques

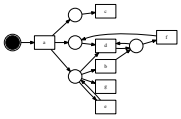
In this section, we introduce the 3 different approaches to the simplification problem that are the main contributions of this work. Figure 3 illustrates these approaches by applying each one to the input model shown in Fig. 2a. The first approach reduces the input to a Petri net that is visually close to a *series-parallel graph* [8] by removing the least important arcs and places (Fig. 2b). However, it has the greatest computational cost. We introduce a second approach that reduces the simplification problem to an Integer Linear Programming (ILP) optimization problem that is more efficient and optionally guarantees the preser-



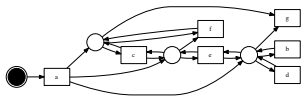
(a) Original model using algorithm in [4].



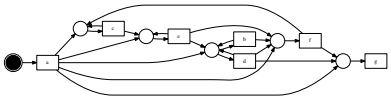
(b) Simplified to series-parallel.



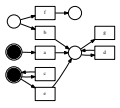
(c) Simplified using ILP model.



(d) Simplified using ILP model (preserving fitness).



(e) Simplified to free choice.



(f) Simplified to state machine.

Fig. 2. Overview of the different simplification techniques.

vation of fitness (Fig. 2c and d). These two techniques require an estimation of the importance of arcs and places. In Section 4 we explain how this scoring is computed, while Section 5 describes the techniques in more detail.

The third technique, however, does not consider any information from the log. Instead, the Petri net is projected into different structural classes: free choice (Fig. 2e) and state machine (2f). This approach is described in Section 6.

4 Log-based Arc Scores

Given a Petri net and an event log, in this section we introduce a technique to obtain a scoring of the arcs (and, indirectly, places) of the net with respect to their importance in describing the behavior underlying in the log.

The idea of the proposed technique is simple: when a Petri net replays a particular trace in the log, some arcs may have more importance than others for that particular trace. Hence, *triggering* and *utilization* scores will be defined to provide an estimation of the importance of the arcs in replaying the log. Arcs $\mathcal{F}(p,t) \neq 0$ with high trigger score correspond to frequent situations in the model where more behavior should not be allowed (i.e., the arc, and therefore p, is frequently disabling certain transitions to occur). By keeping these arcs/places in the model, one aims at deriving a model where precision is not degraded. Conversely, an arc $\mathcal{F}(t,p) \neq 0$ with high utilization score denotes a situation where transition t is frequently fired (thus frequently adding tokens to p), and therefore should not be removed to avoid degrading fitness.

Definition 2 (Trigger Arc). *Let $N = \langle P, T, \mathcal{F}, m_0 \rangle$ be a Petri net, σ a fitting trace for N, and $t' \in \sigma$ a transition represented by firing $m[t'\rangle m'$ in N. For any pair $p \in P$, $t \in T$, an arc $\mathcal{F}(p,t) \neq 0$ is trigger in $m[t'\rangle m'$ iff t is not enabled in m but enabled in m' and $m(p) < \mathcal{F}(p,t)$ but $m'(p) \geq \mathcal{F}(p,t)$.*

Intuitively, an arc $\mathcal{F}(p,t) \neq 0$ is trigger at every transition $t' \in \sigma$ in which t becomes enabled and p is in the set of places which, in that transition t', received the last tokens required for enabling t. Thus, a frequently-trigger arc indicates p is important in restricting the behavior allowed by the model, and that p or $\mathcal{F}(p,t)$ cannot be removed without sacrificing precision. Note that for a single transition t there may be more than one trigger arc, even in the same transition $t' \in \sigma$. To use this information, we define a trigger score which characterizes the frequency of an arc in playing the trigger role. In the following definition, we include the score for arcs between transitions and places, the utilization score, which is based on the frequency of firing:

Definition 3 (Trigger/Utilization Score of an Arc). *Given a Petri net $N = \langle P, T, \mathcal{F}, m_0 \rangle$ and fitting log L, the trigger score of an arc $\mathcal{F}(p,t) \neq 0$, denoted by $\mathcal{T}(p,t)$, is the number of transitions from L in which $\mathcal{F}(p,t)$ is trigger. The utilization score of an arc $\mathcal{F}(t,p) \neq 0$, denoted by $\mathcal{U}(t,p)$, is the number of times transition t is fired in L.*

Given a log and a Petri net, obtaining the trigger/utilization scores can be done by *replaying* all traces in the log. Algorithm 1 shows how to compute trigger

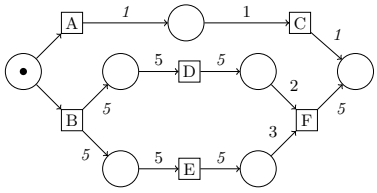
Algorithm 1: TriggerScores

Input: An event log L and a Petri net $N = \langle P, T, \mathcal{F}, m_0 \rangle$
Output: A score $\mathcal{T}(p,t)$ for every arc $\mathcal{F}(p,t) \neq 0$

for $\sigma \in L$ **do**
  Let $m_0[t_1\rangle m_1[t_2\rangle \dots [t_n\rangle m_n = \sigma$
  for $i \leftarrow 1$ **to** n **do**
    for $t \in T$ **do**
      if t *is enabled in* $m_i \wedge t$ *was not enabled in* m_{i-1} **then**
        for $p \in {}^\bullet t$ **do**
          // t is enabled in $m_i \implies m_i(p) \geq \mathcal{F}(p,t)$
          if $m_{i-1}(p) < \mathcal{F}(p,t)$ **then** $\mathcal{T}(p,t) \leftarrow \mathcal{T}(p,t) + 1$
return $\mathcal{T}$

AC
BDEF
BDEF
BDEF
BEDF
BEDF

(a) Example trace



(b) Petri net with trigger/utilization scores

Fig. 3. Trigger and utilization score computation for an example trace and model.

scores: for every transition in the log, the scores are updated by comparing the markings from the predecessor places of all newly enabled transitions.

Figure 3 shows the results of computing, on an example trace and model, both trigger and utilization scores. Utilization scores are shown in *italics*.

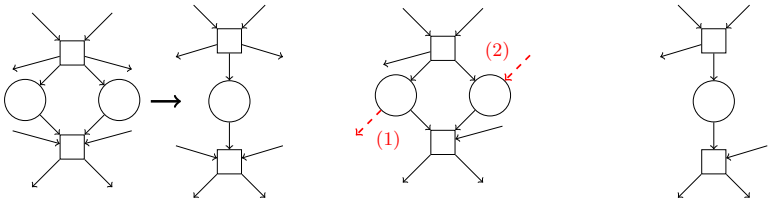
Finally, notice that in the definitions of this section we consider fitting traces. Given an unfitting trace (i.e., a trace that cannot be replayed by the model), an *alignment* between the trace and the model will provide a feasible sequence that is closest to the trace [7]. This allows widening the applicability of the scoring techniques of this section to any pair (log, model).

5 Simplification Techniques using Log-based Scores

5.1 Simplification to a Series-Parallel Net

A series-parallel net is one obtained by the recursive series or parallel composition of smaller nets. Series-parallel Petri nets are amongst the most comprehensible types of models. In a series-parallel net, forks and choices (and thus concurrency) are immediately visible. In fact, existing documentation often uses series-parallel nets as examples to illustrate concepts related to Petri nets.

For this reason, one of the main contributions in this work is a heuristic that reduces a complex Petri net into an almost series-parallel net. The algorithm iteratively removes the least important edges until the graph is either strictly series-parallel, or no additional reduction can be applied without losing the connectedness of the net. The importance of every arc is determined by their trigger score $\mathcal{T}(p,t)$, for place-transition arcs, and their utilization score $\mathcal{U}(t,p)$ for transition-place arcs. The approach is grounded in the notion of a set of reduction rules, explained below.



(a) Reduction rule (b) Source petri net (with rule violations). (c) Transformed net

Fig. 4. Applying a transformation and transformation cost.

Reduction rules. In [5] a set of reduction rules used for the analysis of large Petri net systems is introduced. Each of the transformations preserves liveness, safeness and boundedness of a Petri net. Thus, verification of these properties can be done in the simplified net instead of the original one. The transformations proposed are: *fusion of series places/transitions*, *fusion of parallel places/transitions* and *elimination of self-loop places/transitions*. A rule can be applied only when its preconditions are satisfied. An example of the *fusion of parallel places* rule can be seen in Fig. 4a.

Because of the construction of a series-parallel Petri net, it is possible to reduce such a net to a single place or transition by recursive application of these transformations. Therefore, every violation of the preconditions of a rule indicates a subnet which is not series-parallel.

To reduce a Petri net to a series-parallel skeleton, this work uses these reduction rules in an indirect way. We do not use the transformed Petri net that results from the application of the rules. Instead, the proposed method removes those arcs and places *which prevent the rules from being applied*. For every one of the reduction rules, a *transformation cost* is defined: the sum of the trigger and utilization scores of all the arcs that would need to be removed in order to apply such transformation. The transformation cost therefore models the importance of the arcs that would need to be removed.

Figure 4 shows an example rule, the computation of its transformation cost, and the resulting graph after applying the transformation rule. This rule can only be applied in this input Petri net if two arcs (dashed lines in Fig. 4b) are removed. Thus, the transformation cost is equal to the trigger score of arc (1) and utilization score of arc (2).

Algorithm. Algorithm 2 describes the main iteration of the method. Function `ApplicableTransformations` identifies all possible applications of the reduction rules, and computes the transformation cost for each of the possible applications.

At every iteration we select the transformation m with the least cost, that is, the one that requires removing the least amount of important arcs in order to be applied. Function `ApplyTransformation` applies such transformation m. If applying the transformation breaks the net into more than one connected component, the next best transformation is selected instead. Otherwise, function `PreconditionViolatingArcs` enumerates all the arcs that had to be removed in order to satisfy the preconditions of transformation m. Those arcs are removed them from the original Petri net N_0. The next iteration repeats the process on the transformed net N', finding new `ApplicableTransformations` only around the nodes that were changed on the previous iteration.

Algorithm 2: Series-Parallel algorithm

Input: A Petri net $N_0 = \langle P, T, \mathcal{F}, m_0 \rangle$, a trigger score $\mathcal{T}(p,t)$ for every (p,t) arc, and a utilization score $\mathcal{U}(t,p)$ for every (t,p) arc
Output: A simplifed Petri net

$N \leftarrow N_0$
$M \leftarrow$ `ApplicableTransformations(`N`)`
while $|M| > 0$ **do**
  $m \leftarrow$ transformation with least cost from M
  $N' \leftarrow$ `ApplyTransformation(`N, m`)`
  if N' *is disconnected* **then**
    $M \leftarrow M \setminus \{m\}$
    continue
  $N_0 \leftarrow N_0 \setminus$ `PreconditionViolatingArcs(`N, m`)`
  $N \leftarrow N'$
  $M \leftarrow$ `ApplicableTransformations(`N`)`
return N_0

Once no additional reduction rules can be applied (e.g. because the net is now a single place or transition), the algorithm stops. The currently transformed graph is discarded, and the result of the algorithm is the simplified Petri net N_0. A final postprocessing step removes unneeded places (e.g. without incident arcs).

The nets generated by this heuristic are not necessarily fully series-parallel, since we never remove any arc that would result into an unconnected graph. This is the only method from this work that presents such a global guarantee, with the other methods providing weaker connectivity constraints. It is also possible to configure the method to generate strictly series-parallel models.

5.2 Simplification Using ILP Models

In this section we show a different approach to simplify a Petri net for visualization. The selection of which arcs to remove is seen as an optimization problem, and modeled as an Integer Linear Program (ILP). The use of ILP allows for highly efficient solving strategies. On the other hand, some constraints cannot be modeled using ILP. For example, the models attempt to preserve connectivity of the net at a localized level (i.e. ensuring transitions maintain at least one predecessor and successor place), but cannot guarantee global net connectivity.

The aim of the ILP model is to reduce the number of arcs as much as possible. The inputs are a Petri net $N = \langle P, T, \mathcal{F}, m_0 \rangle$, trigger scores $\mathcal{T}(p,t)$ and utilization scores $\mathcal{U}(t,p)$. We define a binary variable $S(p)$ for every $p \in P$, and a binary variable $A(p,t)$ or $A(t,p)$ for every arc in N. In a solution of this model, variable $S(p)$ is 0 when place p is to be removed from the input graph (similarly for arc variables $A(p,t)$ and $A(t,p)$). Below we describe the ILP model in detail.

The objective function, Eq. 1, minimizes the number of preserved arcs. Constraint 2 encodes the relationship between A and S variables. A place is retained in the output net iff at least one predecessor or successor arc is retained.

The model ensures that the most important arcs, according to the trigger scores $\mathcal{T}$, are preserved. To implement this, constraint 3 imposes a minimum number of preserved arcs: where Γ can be configured as a percentage of the combined trigger score from all place transition arcs. A similar threshold constant Φ is imposed using the utilization score $\mathcal{U}$ for transicion place arcs (Eq. 4).

$$\min \sum_{\mathcal{F}(p,t)>0} A(p,t) + \sum_{\mathcal{F}(t,p)>0} A(t,p) \tag{1}$$

$$\text{s.t. } \forall p \in P : S(p) \iff \sum_{t \in p^\bullet} A(p,t) > 0 \wedge \sum_{t \in {}^\bullet p} A(t,p) > 0 \tag{2}$$

$$\sum_{\mathcal{F}(p,t)>0} \mathcal{T}(p,t) A(p,t) >= \Gamma \tag{3}$$

$$\sum_{\mathcal{F}(t,p)>0} \mathcal{U}(t,p) A(t,p) >= \Phi \tag{4}$$

$$\forall t \in T : \sum_{p \in t^\bullet} A(t,p) > 0 \wedge \sum_{p \in {}^\bullet t} A(p,t) > 0 \tag{5}$$

$$\forall p \in P : M(p) > 0 \implies S(p) \tag{6}$$

$$\forall t \in T, p \in P : \mathcal{F}(t,p) > 0 \wedge S(p) \implies A(t,p) \tag{7}$$

A fully connected graph cannot be guaranteed by the ILP model. Instead, Eq. 5 models a weaker constraint: every transition will preserve at least one predecessor and successor arc. In addition, every place marked in m_0 is always preserved, to avoid deriving a structurally deadlocked model (Eq. 6).

Preserving fitness (optional). The ILP model as described so far does not guarantee preservation of fitness from the original Petri net. A simple modification can ensure that the existing fitness is preserved, at the cost of being able to remove only a reduced number of arcs from the model. Following a well-known result in Petri net theory, removing only $\mathcal{F}(t,p)$ arcs never reduces the fitness of a model for any given log. Constraint 7 implements this restriction.

6 Simplification by Projection into Structural Classes

In this section we present ILP models to reduce Petri nets to two types of structural classes: free choice and state machines [5]. These methods do not require a log as they do not use trigger or utilization scores. Therefore, these proposals can be used to simplify Petri nets for visualization even when logs are not available, albeit their results may be of lower quality since scoring information is not used.

Note that [4] can also be configured to generate state machines or marked graphs, but this approach requires having a log. In addition, the models extracted may still be complex because of the requirement to preserve fitness.

6.1 Free Choice

In this method, we simplify Petri nets by converting them into free choice nets. This method preserves the fitness of the model, but reduces precision. While this reduction does not necessarily result in models simple enough for visualization, complexity is reduced while mantaining most structural properties. Thus, reducing a dense net into free choice both opens the door to efficient analysis and to further decomposition (state machine or marked graph *covers*) techniques [9].

We encode this definition as a set of constraints and create a ILP problem which maximizes the number of arcs. For every $p \in P, t \in T$, we define a binary variable $A(p,t)$ which indicates whether arc $\mathcal{F}(p,t)$ is preserved.

$$\max \quad \sum_{\mathcal{F}(p,t)>0} A(p,t) \tag{8}$$

$$\text{s.t.} \quad \forall p \in P : |p^\bullet| > 1 \wedge |{}^\bullet(p^\bullet)| > 1 \implies \sum_{t \in p^\bullet} A(p,t) = 1 \quad \vee \quad \forall t \in p^\bullet, p' \in {}^\bullet t : p \neq p' \implies \neg A(p',t) \tag{9}$$

Equation 9 guarantees a free choice net. If $|p^\bullet| > 1$ (it is a choice) and ${}^\bullet|p^\bullet| > 1$ (it is not free), then p contains a non-free choice, and one of the conditions must be removed. Either only one of the successor arcs of p is preserved, eliminating the choice, or it is turned free by removing every predecessor arc of $p^\bullet$ except for the ones originating from p itself. Because $\mathcal{F}(t,p)$ arcs are never being removed, this simplification preserves fitness.

6.2 State Machine

In a state machine Petri net, every transition has exactly one predecessor arc and one successor arc. To encode this requirement into an ILP model, we again define a binary variable $A(p,t)$ or $A(t,p)$ for every arc in N.

$$\max \sum_{\mathcal{F}(p,t)>0} A(p,t) + \sum_{\mathcal{F}(t,p)>0} A(t,p) \tag{10}$$

$$\text{s.t. } \forall t \in T : \sum_{\mathcal{F}(p,t)>0} A(p,t) = 1 \tag{11}$$

$$\forall t \in T : \sum_{\mathcal{F}(t,p)>0} A(t,p) = 1 \tag{12}$$

Constraints 11 and 12 encode the definition of a state machine. However, note that this method may reduce the fitness of the model. A similar ILP model can be created to extract a marked graph.

7 Experimental Evaluation

The methods proposed in this work have been implemented in C++. The ILP-based methods have been implemented using a commercial ILP solver, Gurobi [10]. To obtain the input models, the *ILP miner* [4] available in ProM 6.4 was used over a set of 10 complex logs. The publicly available `dot` utility [3] has been used to generate the visualizations of all the models of the paper. The measurements of fitness and precision have been done using alignment-based conformance checking techniques [7]. Both the logs and our implementation are publicly available at `http://www.cs.upc.edu/~jspedro/pnsimpl/`.

7.1 Comparison of the Different Simplification Techniques

In Section 5 (Fig. 3), an artificial model was used to illustrate the different simplification techniques presented in this work. Table 1 shows the details for each one of the simplified models.

Several metrics are used to evaluate the results from the simplification techniques. To evaluate the understandability and simplicity of a model, we use the size of the graph, in number of *nodes* and *arcs*, as well as the number of *crossings*. This is the number of arcs that intersect when the graph is embedded on a plane. Thus, a planar graph has no crossings. A graph with many crossing arcs is clearly a *spaghetti* that is poorly suited for visualization. To approximate the number of crossings, the *mincross* algorithm from `dot` [3] is used.

Table 1. Simplicity, precision and fitness comparison for models in Fig. 3.

	Nodes	Arcs	Crossings	Fitness	Precision
(*a*) Original net	13	35	7	100%	43.1%
(*b*) Series-parallel	13	17	0	100%	37.9%
(*c*) ILP model	12	16	0	68%	75.4%
(*d*) ILP (fitting)	11	21	0	100%	40.7%
(*e*) Free choice	13	24	1	100%	31.3%
(*f*) State machine	13	13	0	49.2%	81.3%

To measure how much the simplified Petri nets model the behavior of the original process we use *fitness* and *precision*, as defined in [7]. In this example, the series-parallel reduction offers perfect fitness, and only 5% loss of precision while removing half of the arcs and all crossings. However, the other methods also remain interesting. For example, the state machine simplification offers the best reduction in simplicity and increases the precision of the model to 80%, at the cost of reducing the fitness by 50%.

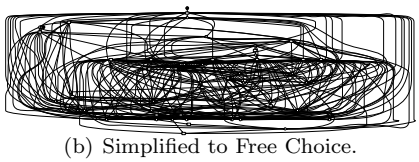
Figure 5 shows the Petri nets produced by the different techniques on a real-life log that is more *spaghetti*-like. The high number of crossings in the original model make it unsuitable for visualization. In this example, the series-parallel method no longer offers perfect fitness but still shows a good trade-off between complexity and fitness/precision. The other methods may be used if for example strict fitness preservation is required, at the cost of more complex models.

In Fig. 6, we compare numerically the techniques of this paper for the 10 logs. For most of the logs, the series-parallel reduction and the ILP-based techniques are able to reduce the number of crossing edges by several orders of magnitude (note the logarithmic scale), creating small visualizable graphs from models that would otherwise be impossible to layout. On the other hand, the simplification to free choice results in very large and complex models. As mentioned, the benefits of deriving free choice models come from the ability to apply additional reduction strategies. Simplifying to state machines generally produces poorly fitting models, but they tend to have very few crossings and high precision.

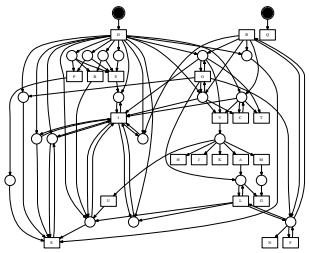
Figure 6 also includes a comparison with some of the previous work in the area: the *Inductive Miner* (IM) [11] and a *unfolding*-based method [2]. The IM is a miner guided towards discovering block-structured models and which we see as a promising technique (see Section 8) since it can be tunned to guarantee perfect fitness. Generally, models generated by the IM contain fewer crossings, caused by the addition of a significant number of *silent* transitions[2] which increase the

[2] A silent activity in the model is not related to any event in the log.

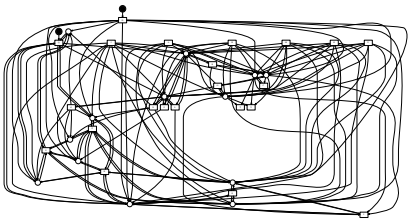
(a) Original Petri net.



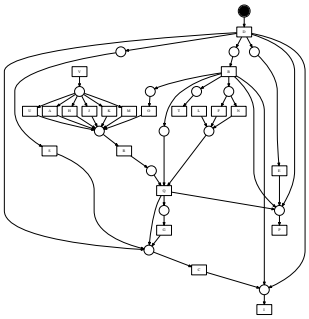
(b) Simplified to Free Choice.



(c) Using ILP model, 60% threshold.



(d) Using ILP model (fitting), 60% threshold.



(e) Simplified to Series-parallel.

	Nodes	Arcs	Crossings	Fitness	Precision
(a)	54	448	9805	100%	31.5%
(b)	54	320	5069	100%	19.4%
(c)	44	93	76	76.7%	42.2%
(d)	37	163	728	100%	15.6%
(e)	39	54	2	74.5%	37.8%

(f) Fitness and precision results.

Fig. 5. Running all methods on real-life log (*incidenttelco*).

size of the model. For example, in the *incidenttelco* example the number of transitions of the model derived by the IM is 37, whilst the original model (and, correspondingly, those generated by the simplification techniques) has 22. The addition of silent activities can be beneficial for visualization, specially if the underlying process model is meant to be block-structured.

On the other hand, the unfolding procedure is more closely related to the methods proposed in this work. This technique uses an *unfolding* process to simplify an existing Petri net, and has been evaluated using the same nets as with our proposed methods. In general, it produces better results in terms of fitness and precision with respect to the ILP models, at the expense of longer computation time. When compared with the series-parallel method, the results in fitness and precision are comparable, but the unfolding method requires more computation time and the results are worst in terms of visualization.

In Fig. 7 we compare the runtimes of the different methods. The ILP solver resolved all the ILP simplification models in less than 1 minute, even for the largest of the input Petri nets from the test set (25K nodes and arcs). The series-parallel simplification, which is not ILP based, has a lower performance.

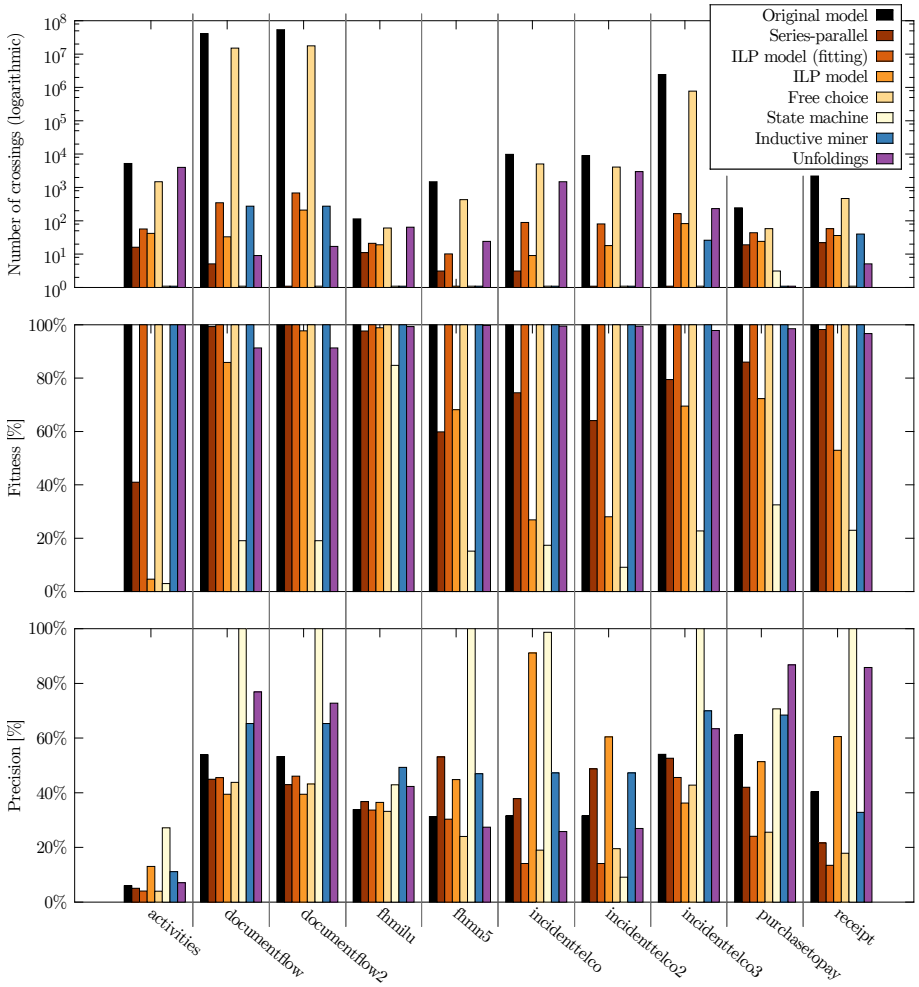


Fig. 6. Simplicity (logarithmic scale on the number of crossings), fitness and precision comparison between the different techniques using 10 real-life logs.

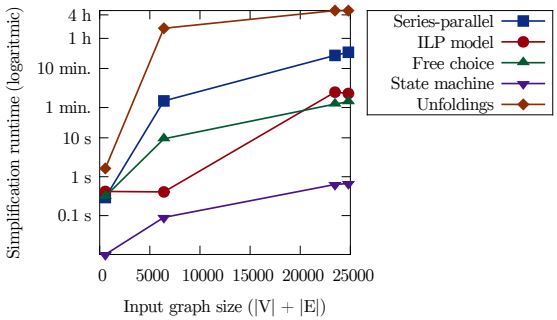


Fig. 7. Execution runtimes for the different simplification techniques.

However, there are many parts where the algorithm could be optimized. Still, the total execution runtime for the largest graph (25 minutes) was less than the 1 hour required for the miner in [4] to generate the input Petri net from the log, and significantly less than the 5 hours required by the unfolding technique presented in [2] (also shown in the plot).

The experiments presented in this section show the proposed simplification ILP models to be highly efficient and able to generate models that are orders of magnitude simpler than the original models. If additional simplification is required, the series-parallel method can be used with an increased runtime.

7.2 Effect of the Threshold Parameter on the ILP Model

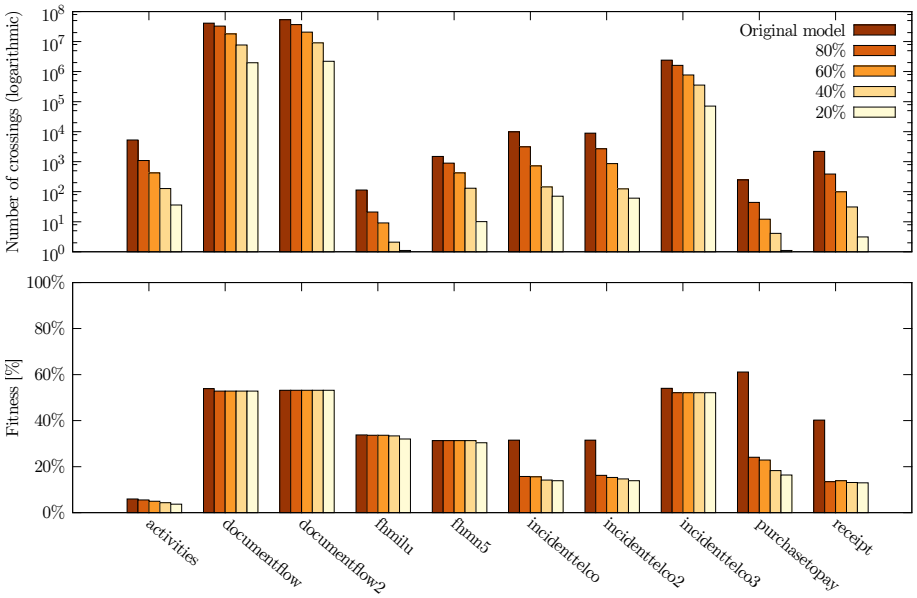


Fig. 8. Simplicity and fitness comparison using different thresholds for the ILP model.

The ILP simplification model presented in Section 5.2 contains a threshold parameter (Γ and Φ) which can be used to tune the complexity and size of the simplified models. In previous experiments and figures, a threshold was set manually so that models with approximately $2|T|$ arcs were generated (where T is the set of transitions from the input Petri net).

To illustrate how varying these thresholds affects the model complexity and quality, the ILP simplification model was executed for each of the input logs, with varying threshold parameters. Fig. 8 shows the number of crossings and the fitness for each combination. Generally the fitness decreases with the threshold parameter, but there are some models where the trend reverses. This is because nothing in the model ensures that a log with a given threshold Γ will strictly capture all the behavior of a log simplified using Γ' with $\Gamma > \Gamma'$.

8 Related Work

The closest work to the methods of this paper is [2], where a technique was presented for the simplification of process models that controls the degree of precision and generalization. It applies several stages. First, a log-based unfolding of the model is computed, deriving a precise unfolded model. Second, this unfolding is then filtered, retainning only the frequent parts. Finally, a folding technique is applied which controls the generalization of the final model. Further simplifications can be applied, which help on alleviating the complexity of the derived model. There are significant differences between the two approaches: while in our case, the techniques rely on light methods and can be oriented towards different objectives, the approach in [2] requires the computation of unfoldings, which can be exponential on the size of the initial model [12]. Also, the filtering on the unfolding is done on simple frequency selection on the unfolding elements, while in this paper the importance of model elements is assessed with the frequency but also triggering information, which is related to the precision dimension. On the other hand, the techniques of this paper may need to verify model connectedness at each iteration. In conclusion, both techniques can be combined to further improve the overall simplification of a model.

The simplification of a process model should be done with respect to quality metrics, and in this paper we have focused on fitness and precision. An alternative to this approach would be to include these quality metrics in the discovery, a feature that has only been considered in the past by the family of genetic algorithms for process discovery [13,14,15]. All these techniques include costly evaluations of the metrics in the search for an optimal process model, in order to discard intermediate solutions that are not promising. This makes these approaches extremely inefficient in terms of computing time.

Furthermore, there exist discovery techniques that focus on the most frequent paths [16,17]. These approaches are meant to be resilient to noise, but on the other hand give no guarantees on the quality of the derived model. Additionally, these approaches are oriented towards less expressive models, which makes the simplification task easier than the one considered in this paper. A recent technique that is guided towards the discovery of block-structured models (*process trees*) and that addresses these issues may be a promising direction [11]. However, this technique is guided towards a particular class of Petri nets (workflow and sound), describing a very restricted type of behaviors. Finally, the techniques of this paper can be combined with abstraction mechanisms to further improve the visualization of the underlying process model.

9 Conclusions

A collection of techniques for the simplification of process models using log-based information has been presented in this paper. The techniques proposed tend to improve significantly the visualization of a process model while retaining its main qualities in relation with an event log. This contribution may be used on the model derived by any discovery technique, as an intermediate step between discovery and visualization. Also, the analysis of simplified models may

be considerably alleviated (e.g., if deriving a free-choice net). The experiments done on dense models have also shown a significant simplification capability in terms of visualization metrics like density or edge-crossings.

Acknowledgments. This work as been partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant COMMAS (ref. TIN2013-46181-C2-1-R) and by a grant from Generalitat de Catalunya (FI-DGR). We would also like to thank to Seppe vanden Broucke, Jorge Munoz-Gama and Thomas Gschwind for their great help in the experiments.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Fahland, D., van der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Information Systems **38**(4) (2013) 585–605
3. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.: A technique for drawing directed graphs. IEEE Trans. Software Eng. **19**(3) (1993) 214–230
4. van der Werf, J., van Dongen, B., Hurkens, C., Serebrenik, A.: Process discovery using integer linear programming. In van Hee, K., Valk, R., eds.: Applications and Theory of Petri Nets. Volume 5062 of Lecture Notes in Computer Science., Springer-Verlag (2008) 368–387
5. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (April 1989) 541–574
6. Munoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In Hull, R., Mendling, J., Tai, S., eds.: Business Process Management. Volume 6336 of Lecture Notes in Computer Science. (2010) 211–226
7. Adriansyah, A.: Aligning observed and modeled behavior. PhD thesis, Technische Universiteit Eindhoven (2014)
8. Valdes, J., Tarjan, R.E., Lawler, E.L.: The recognition of series parallel digraphs. SIAM J. Comput. **11**(2) (1982) 298–313
9. Desel, J., Esparza, J.: Free choice Petri nets. **40** (1995)
10. Gurobi Optimization: Gurobi Optimizer reference manual (2015)
11. Leemans, S., Fahland, D., van der Aalst, W.: Discovering block-structured process models from incomplete event logs. In: Application and Theory of Petri Nets and Concurrency. Volume 8489 of Lecture Notes in Computer Science. (2014) 91–110
12. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Computer Aided Verification. Volume 663 of Lecture Notes in Computer Science. (1993) 164–177
13. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In Ciardo, G., Darondeau, P., eds.: Applications and Theory of Petri Nets 2005. Volume 3536 of Lecture Notes in Computer Science. (2005) 48–69
14. Buijs, J.: Flexible Evolutionary Algorithms for Mining Structured Process Models. PhD thesis, Technische Universiteit Eindhoven (2014)
15. Vázquez-Barreiros, B., Mucientes, M., Lama, M.: ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. Information Sciences **294** (2015) 315–333
16. Günther, C.: Process Mining in Flexible Environments. PhD thesis, Technische Universiteit Eindhoven (2009)
17. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: Computational Intelligence and Data Mining (CIDM). (2011) 310–317