P^3-Folder: Optimal Model Simplification for Improving Accuracy in Process Performance Prediction

Arik Senderovich[1], Alexander Shleyfman[1], Matthias Weidlich[2], Avigdor Gal[1], and Avishai Mandelbaum[1]

[1] Technion – Israel Institute of Technology
{sariks@tx,alesh@tx,avigal@ie,avim@ie}.technion.ac.il
[2] Humboldt-Universität zu Berlin, Berlin, Germany
matthias.weidlich@hu-berlin.de

Abstract. Operational process models such as generalised stochastic Petri nets (GSPNs) are useful when answering performance queries on business processes (e.g. 'how long will it take for a case to finish?'). Recently, methods for process mining have been developed to discover and enrich operational models based on a log of recorded executions of processes, which enables evidence-based process analysis. To avoid a bias due to infrequent execution paths, discovery algorithms strive for a balance between over-fitting and under-fitting regarding the originating log. However, state-of-the-art discovery algorithms address this balance solely for the control-flow dimension, neglecting possible over-fitting in terms of performance annotations. In this work, we thus offer a technique for performance-driven model reduction of GSPNs, using structural simplification rules. Each rule induces an error in performance estimates with respect to the original model. However, we show that this error is bounded and that the reduction in model parameters incurred by the simplification rules increases the accuracy of process performance prediction. We further show how to find an optimal sequence of applying simplification rules to obtain a minimal model under a given error budget for the performance estimates. We evaluate the approach with a real-world case in the healthcare domain, showing that model simplification indeed yields significant improvements in time prediction accuracy.

1 Introduction

Performance analysis is an important pillar of business process management initiatives in diverse domains, reaching from telecommunication, through healthcare, to finance. Taking healthcare as an example, it involves the ability to answer questions such as 'how long will it take for a patient to get treatment?', and 'how many nurses do we need to staff to accommodate the incoming demand?'. Answers to these questions are key in running an organization successfully and deliver value to its clients [1].

Operational process models such as generalised stochastic Petri nets and queueing networks are useful in answering the aforementioned performance questions [2,3]. In particular, these models enable testing of re-design and improvement initiatives with respect to the as-is model. For instance, by changing staffing levels and altering the control-flow, the impact of operational changes on the performance characteristics of the process can be explored.

Process mining enables automatic discovery and enrichment of operational process models from logs, which record process executions [4]. Data-driven model discovery improves beyond the manual model elicitation in its ability to reflect the process as it is actually executed. However, automatically discovered models tend to incorporate infrequent process executions, which may result in over-fitting with respect to the originating log. Recently proposed discovery algorithms attempt to balance between over-fitting and under-fitting in the control-flow dimension [5,6,36,7]. Yet, the question of how to avoid over-fitting in terms of performance annotations of operational models has not been addressed in the literature.

This work approaches the problem of over-fitting of operational process models with P^3-Folder, a method for automated simplification of generalised stochastic Petri nets (GSPNs) for process performance prediction. Starting with an over-fitting GSPN discovered from a log, the idea behind P^3-Folder is that simplification reduces the number of model parameters. This, in turn, increases the accuracy of performance estimates, even though simplification generalises the model by introducing an estimation error regarding the original model. More specifically, our contribution is twofold. As a first step, P^3-Folder defines a set of structural simplification rules for GSPNs, referred to as *foldings*. Unlike existing proposals for model simplification [8], these rules are local (affecting only a subnet of the GSPN), come with formal bounds regarding the introduced estimation error, and their applicability is identified automatically by structural decomposition of the GSPN. Second, P^3-Folder formulates model simplification as an optimization problem that aims at attaining a minimal model for a given budget for the introduced estimation error. This problem is cast as an Integer Linear Programming (ILP) problem, which enables efficient computation of the optimal sequence of folding operations.

We evaluate P^3-Folder with a case in the healthcare domain. Our experiments show that simplification of a GSPN discovered from a real-world log yields a significant improvement in time prediction accuracy compared to the original GSPN.

The remainder of the paper is structured as follows. The next section discusses the methods and challenges in performance-oriented process mining. Section 3 recalls the GSPN formalism. Foldings of GSPNs are introduced in Section 4. The model simplification problem and its encoding as an ILP program is proposed in Section 5. Evaluation results are presented in Section 6. Section 7 reviews related work, before Section 8 concludes the paper.

2 Background: Performance-Oriented Process Mining

Process Mining for Operational Analysis. We consider a setting in which a log L of recorded process executions is given and analysis questions regarding the performance of process execution shall be answered. Specifically, let Y be a performance measure, e.g., the total runtime of a process instance. Further, let $q(Y)$ be a performance query over Y, e.g., the expected value of Y, which we aim at answering based on L. In general, we distinguish two types of process mining techniques to quantify $q(Y)$.

First, machine learning (ML) techniques may be exploited. That is, process executions (including their data) are encoded as a feature vector X. Common ML methods such as regression or decision trees are used to construct an estimator $\hat{q}(Y)$ conditioned on X. Examples for such methods are found in [9,10,11]. While such an approach is often accurate in predicting $q(Y)$, it has two major drawbacks. Given a performance measure Z that is not directly observable in the log, one needs to quantify $q(Z)$, since ML methods require labelled observations of $q(Z)$ in the training phase. For instance, Z may be the waiting time for a specific resource. If it is not recorded in the log, $q(Z)$ must be estimated. This estimation procedure may introduce an error, which will reduce the accuracy of the learning technique. In addition, exploring to-be processes and sensitivity analysis of current process parameters is impossible due to lack of data that describes the effect of X on Y under the new terms.

A second angle to answer performance question is to use operational process models. Given the log L, operational models such as GSPNs can automatically be discovered and enriched with performance information [12,13]. To quantify $q(Y)$, a corresponding query $q_M(Y)$ is evaluated over the model, e.g., with the help of simulation [13] or queueing theory approximations [8,3]. A model-based approach overcomes the aforementioned limitations. It supports queries for measures that were not directly recorded in the log and enables to-be performance analyses and sensitivity analysis (e.g., by changing the control-flow and altering activity durations).

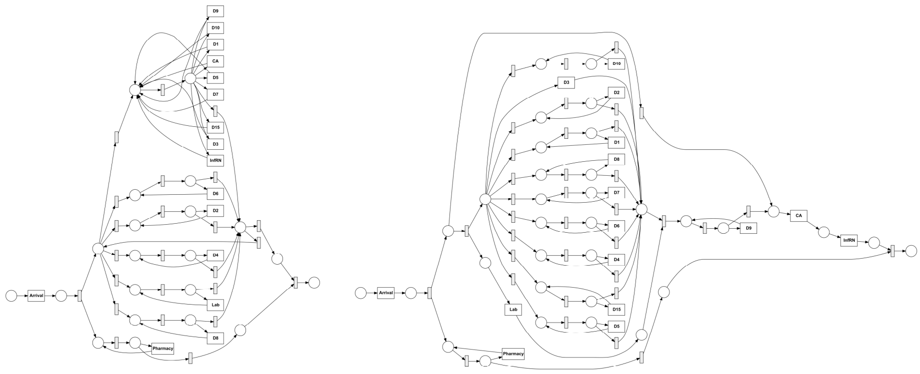
However, a model-based approach also suffers from a major drawback, namely *over-fitting* of the estimated $q(Y)$ with respect to L [8]. ML-based methods balance over-fitting of $\hat{q}$ to L by means of regularization methods (e.g., pruning the regression tree [10]). A model-based approach for regularizing, in turn, does not exist.

We illustrate this problem with two models that were discovered from a real-world case in the healthcare domain (see our evaluation results for details). Fig. 1 depicts the two process models discovered using the Inductive Miner [14] with different noise thresholds: 0% for model (a) and 20% for model (b). We observe that noise filtering balances over- and under-fitting of the control-flow regarding the log, yielding a more sequential model when filtering more noise events. However, the trade-off between over- and under-fitting is not addressed for the performance perspective. Enriching both models with performance information based on [12] and testing them against a month of operational data not used in model construction shows that model (b) is only slightly more accurate than model (a). As we later demonstrate experimentally, a principled approach based on model simplification, in turn, alleviates over-fitting in the performance dimension, thereby significantly improving prediction accuracy.

Resolving Performance Overfitting by Model Simplification. The idea followed in this work is to avoid over-fitting in the performance dimension by model simplification. We balance model size (number of model parameters) and proximity of the performance estimates of the simplified model to those of the original model (and thus the log).

To explain this idea in more detail, we adopt a statistical perspective on discovery and enrichment of operational process models. We assume that the performance measure Y is governed by a (parametric and stochastic) process model M, i.e., $Y \sim M$. Then, an estimation of $q(Y)$ translates into an estimation of $q(M)$. Hence, it is sufficient to estimate the model M to obtain $q(Y)$.

Common process discovery and enrichment techniques yield an initial model M_0 of the model parameters. Then, our P^3-Folder method applies a sequence of simplification rules to M_0, each introducing an estimation error with respect to $q(Y)$. What prevents



(a) 0% noise filter. (b) 20% noise filter.

Fig. 1: Automatically discovered model of a hospital process.

the model from collapsing into a single node is an error budget B. P^3-Folder generates a model M^* as the most simple one in terms of size that is still with the specified error bound from M_0 with respect to $q(Y)$. Tuning the error budget B for a specific measure Y is performed via a cross-validation procedure on L. The resulting model M^* 'enjoys' the benefits of a model-based performance analysis, while being more general compared to M_0. As such, P^3-Folder can be viewed as a regularization of process models, transferring the analogy of machine learning into the world of model-based performance analysis.

3 Performance Analysis with Generalised Stochastic Petri Nets

GSPN Syntax and Semantics. Generalised Stochastic Petri Nets (GSPNs) [15] are a class of Petri nets that incorporate stochastic information on time behaviour: transitions are either *immediate*, representing atomic logical actions, or *timed*, representing units of work. Below, we recall a notion of GSPNs that includes weights of immediate transitions, and resource capacities and expected durations of timed transitions.

Definition 1 (GSPN)**.** A GSPN is a tuple $G = \langle P, T, F, \gamma, \delta, \omega \rangle$ where:
- P is the set of places,
- $T = T_i \cup T_t$ is the set of transitions consisting of immediate transitions T_i and timed transitions T_t, respectively,
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation,
- $\gamma : T_t \to \mathbb{R}_0^+$ assigns capacities to timed transitions (work units per time unit).
- $\delta : T_t \to \mathbb{R}_0^+$ assigns expected durations to timed transitions.
- $\omega : T_i \to [0, 1]$ assigns weights to immediate transitions.

We refer to the tuple $\langle P, T, F \rangle$ as the *structure* of the GSPN, and to $\langle \gamma, \delta, \omega \rangle$ as its *functional component*. The set $X = P \cup T$ denotes all *nodes* and the *size* of a GSPN is defined as $|X|$. For a node x, $\bullet x = \{y \in X \mid (y, x) \in F\}$ and $x\bullet = \{y \in X \mid (x, y) \in F\}$ denote its *preset* and *postset*, respectively. Further, F^* is the transitive closure of F.

Semantics of a GSPN are defined as a 'token game': A *marking* $M : P \to \mathbb{N}_0$ assigns to each place a number of *tokens*, thereby representing a GSPN state. A transition $t \in T$ is *enabled* in M, if all places in its preset are marked, i.e., $\forall\, p \in \bullet t : M(p) > 0$.

An immediate transition that is enabled, can *fire*. Firing of a timed transition t depends on its capacity and expected duration: Once it is enabled, a single exponential clock with rate $\lambda(t) = \frac{\gamma(t)}{\delta(t)}$ is started and the transition can *fire* when the clock is elapsed. That is, we assume a single-server semantics: there is one exponential clock per enabling.

Firing a transition t in a marking M yields a marking M', such that $M'(p) = M(p) - 1$ for all $p \in \bullet t \setminus t\bullet$; $M'(p) = M(p) + 1$ for all $p \in t\bullet \setminus \bullet t$; and $M'(p) = M(p)$ otherwise. Although tokens are indistinguishable, for performance analysis, we shall assume that the tokens that enable a timed transition are selected on a First-Come First-Served (FCFS) policy. Since first-order performance measures (e.g., average waiting times and average number of tokens in a place) are indifferent to the selection policy [15], the assumed FCFS policy is indeed plausible.

Semantics of a GSPN further depend on types of transitions and their assigned rates (capacity over expected duration) and weights as follows. Let $t_1, ..., t_k \in T$ be transitions that are enabled in a marking M, i.e., they compete for firing. If transitions $t_1, ..., t_k$ are either all immediate or all timed, the assigned rates or weights determine the likelihood of each of the transitions being fired. This likelihood is defined for transition t_j, $1 \leq j \leq k$, as $\frac{\lambda(t_j)}{\sum_{i=1}^{k} \lambda(t_i)}$ (only timed transitions) or $\frac{\omega(t_j)}{\sum_{i=1}^{k} \omega(t_i)}$ (only immediate transitions), respectively. If some transitions are immediate and some are timed, the immediate transitions have priority and the likelihood model is applied only to the immediate transitions.

Process Performance Analysis. A business process is described by an *open GSPN*, which is a GSPN $G = \langle P, T, F, \gamma, \delta, \omega \rangle$ that has a dedicated timed transition $\tau_0 \in T_t$, called *arrival transition*, which represents external arrivals into the system [16]. Specifically, it holds that $\bullet\tau_0 = \emptyset$ (and for all $t \in T_t \setminus \{\tau_0\}$ it holds $\bullet t \neq \emptyset$), $\gamma(\tau_0) = 1$, and $\delta(\tau_0) = \frac{1}{\beta_0}$, so that β_0 represents the *arrival rate* of the open GSPN. In the remainder, we assume all GSPNs to be open GSPNs.

The arrival transition τ_0 is enabled in any marking and thus, also in the *empty marking* M_0 with $M_0(p) = 0$ for all $p \in P$, which serves as the *initial marking*. Then, the *reachability graph* of G is a graph comprising all *reachable markings*, denoted $\mathcal{R}(M_0)$, i.e., markings that can be obtained by firing of transitions of G, starting in M_0 (here, the empty marking). To perform steady-state analysis, it was shown that the reachability graph of a GSPN is isomorphic (after reduction) to a Continuous-Time Markov Chain (CTMC) [15]. The transition rates between the CTMC states correspond to the rates $\lambda(t) = \frac{\gamma(t)}{\delta(t)}$ assigned to the respective timed transitions in the GSPN. Exploiting this transformation, performance analysis of a GSPN is based on techniques of CTMC analysis: global balance equations of the CTMC are solved or, to alleviate the complexity of solving these equations, queueing theory approximations can be used. In this work, we use such an approximation technique presented in [17].

4 Foldings of GSPN

P^3-Folder employs folding operations (aka foldings) to simplify GSPNs. We first elaborate on the general notion of foldings, before providing a detailed discussion of an exemplary folding. Finally, we show how to identify applicable foldings based on structural decomposition of a GSPN.

4.1 The Notion of a Folding

A folding operation is a contraction of a GSPN, which yields a GSPN that is equal or smaller in size. Yet, not all contractions are reasonable when aiming at improved accuracy of performance prediction. It is important that foldings preserve *stability* to ensure that the resulting model has a finite expected waiting time value. In GSPN terminology, a timed transition $t \in T_t$ of a GSPN $G = \langle P, T, F, \gamma, \delta, \omega \rangle$ is *stable*, if the marking $M_h(p) = 0$ $, \forall\, p \in \bullet t$ is a home marking for M_0 in G, i.e., $\forall\, M' \in \mathcal{R}(M_0) : M_h \in \mathcal{R}(M')$. We call G *stable*, if all its timed transitions T_t are stable.

Let $\mathcal{G}$ be the universe of GSPNs. Then, we define foldings as follows:

Definition 2 (Folding)**.** A folding is a function $\psi : \mathcal{G} \to \mathcal{G}$, such that for all $G \in \mathcal{G}$ it holds that $|\psi(G)| \leq |G|$. A folding ψ is called *proper*, if for all $G \in \mathcal{G}$ it holds that G being stable implies that $\psi(G)$ is stable.

The preservation of stability, termed *properness*, can be seen as a correctness criterion for the definition of foldings. Aiming at a contraction of the original GSPN, however, most foldings are actually abstractions that imply a certain bias in any performance analysis done with the resulting model. To control the application of foldings, therefore, we assign each folding a *cost* that bounds the possible estimation error. Clearly, this cost is specific to a particular performance measure and, thus, the type of performance analysis that shall be conducted with the folded model. As a prominent example measure, we consider the sojourn time of a GSPN: the total time it takes for the tokens produced by a single firing of the arrival transition τ_0 to reach a deadlocking marking (a marking in which no transition is enabled).

Let G be a GSPN and $G' = \psi(G)$ for some folding ψ. Let S and S' be random variables for the sojourn times of G, and G', respectively. The cost of applying folding ψ to G is defined as the absolute deviation in expectation between the sojourn times: $c(G, \psi) = |\mathbb{E}S' - \mathbb{E}S|$. Note that, since firing delays are given in the GSPN, the main challenge in evaluating sojourn times is obtaining good estimates for waiting times.

In this work, we consider five foldings: (1) sequence-folding, (2) race-folding (3) XOR-folding, (4) AND-folding, and (5) loop-folding. These foldings relate to common behavioural structures in business process models [18]. Each of them yields a simple GSPN comprising the arrival transition and a second timed transitions, as illustrated in Fig. 2. Note that the race-folding and XOR-folding relate to different semantic concepts: the former folds a net that represents resources working in parallel on jobs that arrive as tokens in the respective place. The XOR-folding, in turn, relates to probabilistic selection of activities, i.e., a probabilistic selection among different timed transitions.

All the five foldings are proper and their costs can be computed by exploiting results from queueing theory. Due to space limitations, however, we limit the discussion of properness and costs in this work to XOR-folding.

4.2 The XOR-folding

The XOR-folding, denoted by ψ_X, takes as input a GSPN $G = \langle P, T, F, \gamma, \delta, \omega \rangle$ of the structure visualised in Fig. 2 (D): it comprises the τ_0 transition (with rate β_0), a single place p_i with $\bullet p_i = \{\tau_0\}$ and a single place p_o with $p_o \bullet = \emptyset$ that are connected by sequential structures, each comprising two immediate transitions and a timed transition.

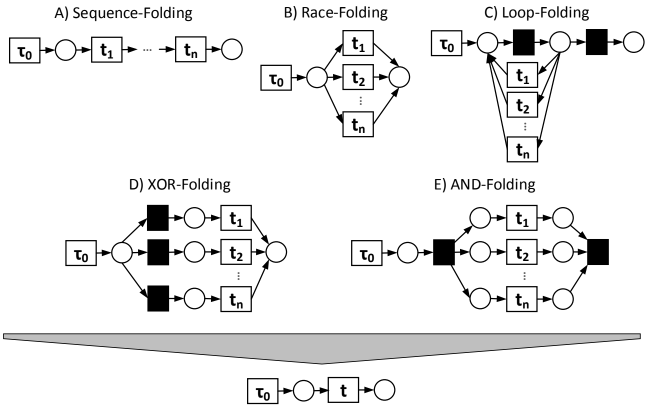


Fig. 2: Overview of foldings.

Applying the folding yields a GSPN $G' = \psi_X(G) = \langle P', T', F', \gamma', \delta', \omega' \rangle$, where the structure $\langle P', T', F' \rangle$ is a trivial net comprising the τ_0 transition ($\gamma'(\tau_0) = \gamma(\tau_0)$ and $\delta'(\tau_0) = \delta(\tau_0)$), and two places that are connected via a timed transition, t, see Fig. 2.

The functional part of G', that is $\langle \gamma', \delta', \omega' \rangle$, is constructed as follows. First, weights (ω') become irrelevant, since G' does not contain immediate transitions. The capacity (γ) and expected duration (δ) of the timed transition t of G' are set as:

- $\gamma'(t) = \sum_{t_t \,\in\, T_t \,\setminus\, \{\tau_0\}} \gamma(t_t)$, i.e., the new transition is allocated the total capacity of the internal timed transitions in G;
- $\delta'(t) = \sum_{t_t \,\in\, T_t \,\setminus\, \{\tau_0\},\, t_i \,\in\, T_i,\, (t_i, t_t) \,\in\, F^*} w(t_i)\delta(t_t)$, i.e., the new transition is assigned an expected duration that is the weighted average of the durations of the timed transitions in G, where the weights stem from the respective immediate transitions.

Theorem 1 ascertains the XOR-folding properness.

Theorem 1. *If G is stable, then $\psi_X(G)$ is stable.*

Proof. By [20], the stability condition for G is that for all $t_t \in T_t \setminus \{\tau_0\}$ it holds that $\beta_0 w(t_i) < \frac{\gamma(t_t)}{\delta(t_t)}$ with $t_i \in T_i$ such that $(t_i, t_t) \in F^*$. Hence, the sum of these inequalities yields $\beta_0 < \sum_{t_t \,\in\, T_t \,\setminus\, \{\tau_0\}} \frac{\gamma(t_t)}{\delta(t_t)}$. Due to

$$\sum_{i=1}^{n} \frac{a_i}{b_i} < \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} b_i},$$

for $a_i, b_i > 0$, we arrive at

$$\sum_{t_t \in T_t} \frac{\gamma(t_t)}{\delta(t_t)} < \frac{\sum_{t_t \in T_t \setminus \{\tau_0\}} \gamma(t_t)}{\sum_{t_t \,\in\, T_t \,\setminus\, \{\tau_0\},\, t_i \,\in\, T_i,\, (t_i, t_t) \,\in\, F^*} w(t_i)\delta(t_t)},$$

which proves stability of G'. □

To calculate the cost of the XOR-folding, we compute the expected sojourn times, S_X in G, and S'_X in $G' = \psi_X(G)$. Since arrivals into the systems (by firing the arrival transition τ_0) are Poisson arrivals, the arrival of timed transitions $t_t \in T_t \setminus \{\tau_0\}$ in G are also Poisson (due to the 'Poisson splitting' property [27]). The arrival rate for

$t_t \in T_t \setminus \{\tau_0\}$ is given as $w(t_i)\beta_0$ with $t_i \in T_i$ such that $(t_i, t_t) \in F^*$. Note that for GSPNs showing concurrency, the 'Poisson splitting' property does not hold true and a refinement of the above approximation can be made by using Eq. (24) in [17].

The firing delays for each of the timed transitions, $t_t \in T_t \setminus \{\tau_0\}$ are assumed to be independent of the arrival process, and have exponential durations. These assumptions enable the use of the $M/M/1$ formula for each timed transition to calculate the sojourn times [16]. We write the expected value of S_X as:

$$\mathbb{E}S_X = \sum_{t_t \in T_t \setminus \{\tau_0\}, t_i \in T_i, (t_i, t_t) \in F^*} w(t_i)\mathbb{E}S_X^{t_t}, \tag{1}$$

Since it is known that

$$\mathbb{E}S_X^{t_t} = \frac{1}{\lambda(t_t) - w(t_i)\beta_0} = \frac{\delta(t_t)}{\gamma(t_t) - \beta_0 w(t_i)\delta(t_t)}, \tag{2}$$

for $t_t \in T_t \setminus \{\tau_0\}, t_i \in T_i, (t_i, t_t) \in F^*$, see [16], the sojourn time is given by:

$$\mathbb{E}S_X = \sum_{t_t \in T_t \setminus \{\tau_0\}, t_i \in T_i, (t_i, t_t) \in F^*} \frac{w(t_i)\delta(t_t)}{\gamma(t_t) - \beta_0 w(t_i)\delta(t_t)}. \tag{3}$$

We now turn to the calculation of the sojourn time S'_X for $G' = \psi_X(G)$:

$$\mathbb{E}S'_X = \frac{1}{\lambda'(t) - \beta_0}, \tag{4}$$

with $\lambda'(t) = \frac{\gamma'(t)}{\delta'(t)}$. In primitives of G, the expected sojourn time is given as:

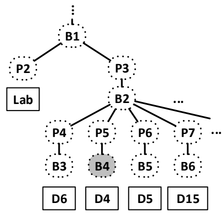
$$\mathbb{E}S'_X = \frac{\sum_{t_t \in T_t \setminus \{\tau_0\}, t_i \in T_i, (t_i, t_t) \in F^*} w(t_i)\delta(t_t)}{\sum_{t_t \in T_t \setminus \{\tau_0\}, t_i \in T_i, (t_i, t_t) \in F^*} \gamma(t_t) - \beta_0 w(t_i)\delta(t_t)}. \tag{5}$$

The resulting cost for the XOR-folding is: $c(G, \psi) = |\mathbb{E}S'_X - \mathbb{E}S_X|$, which is easy to compute as it comprises only of primitives of G, the originating GSPN.

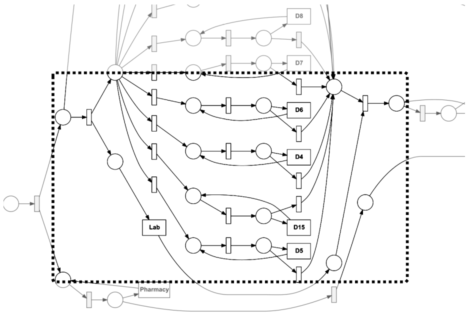
4.3 Finding Foldings by GSPN Decomposition

So far, we discussed the foldings shown in Fig. 2 as a transformation of a complete GSPN of the according structure. However, P^3-Folder employs foldings also to transform parts (aka subnets) of a GSPN, which may enable iterative application of foldings. This holds in particular, as the foldings can be applied to any part of a GSPN that has one of the structures shown in Fig. 2, when removing the arrival transitions τ_0. The reason is that the rate of token arrival into the structures, as encoded by the arrival transitions, can be precomputed by solving the (linear) 'traffic equations' [21,17], which tie the external arrival rate of the entire GSPN to the internal arrival rates of places of the GSPN.

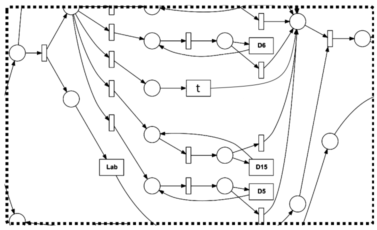
Observing that the structures in Fig. 2, once the arrival transitions have been removed, correspond to common *single-entry/single-exit (SESE)* controlflow structures, P^3-Folder employs structural decomposition of the GSPN to identify applicable foldings. Specifically, the Refined Process Structure Tree (RPST) [22,23] is used to parse a GSPN into a hierarchy of SESE *fragments*. Then, the RPST is a containment hierarchy of *canonical*



(b) Partial RPST



(a) GSPN (timed transitions are labelled)



(c) GSPN part after folding B4

Fig. 3: Example for the decomposition of a GSPN using the RPST.

fragments of the graph, which is unique and can be computed in linear time [23]. Fragments can be classified to one out of four structural classes: trivial fragments consists of a single edge; polygons (P) that are sequences of fragments; bonds (B) that are collections of fragments that share entry and exit nodes; and rigids representing any other structure.

To identify which of the foldings outlined in Fig. 2 can be applied to a given GSPN, we rely on the RPST of the GSPN as follows:

Sequence-Folding: The folding can be applied to any polygon fragment that has only trivial fragments as children and comprises at least two timed transitions. Assuming that the GSPN has been normalised (immediate transitions may occur only as the first child, as they are redundant at any other position in a polygon), the folding applies to the maximal sequence of timed transitions.

Race-Folding/XOR-Folding/AND-Folding: The foldings apply to place-bordered (Race, XOR) or transition-bordered (AND) bond fragments that contain only polygons of single timed transitions (Race, AND) or polygons of three children, a timed transitions that is preceded and succeeded by immediate transitions (XOR).

Loop-Folding: The folding applies to place-bordered bonds that are cyclic and part of a polygon. The bonds needs to be followed by an immediate transition in the parent polygon and show the structure visualised in Fig. 2. That is, the children of the bond are polygons of single transitions that are either immediate (if flows in the child lead from the bond entry to the bond exit) or timed (otherwise).

The above rules identify foldings iteratively: whenever an applicable folding was found, the respective part of the GSPN is replaced by a timed transition and the rules are checked again. This way, given a GSPN, P^3-Folder obtains a set of *folding instantiations* $F = \{f_1, \ldots, f_n\}$, each being defined by a folding and the GSPN that is folded. Further, a precedence function $\nu : F \to (\wp(F) \cup \emptyset)$ defines, given $f \in F$, the set of all folding instantiations that must be applied before f, to generate the GSPN on which f is applied.

We illustrate this approach with a GSPN derived by annotating the model of Fig. 1b. Fig. 3a shows an excerpt of this model. The RPST of the highlighted part is depicted in Fig. 3b. Here, loop-foldings can be applied to all the bond fragments $B3$-$B6$, since they comprise polygons of single transitions of the required types. Applying the loop-folding to bond $B4$ yields the net partially shown in Fig. 3c. Once loop-foldings have been applied to bonds $B3$-$B6$, an XOR-folding can be applied to bond $B2$, which now comprises polygons, each built of an immediate transition followed by a timed transition.

5 Optimal Simplification of GSPN

Using the foldings proposed above, this section shows how P^3-Folder identifies the cost-optimal sequence of foldings to simplify a given GSPN. To this end, we define the problem of optimal folding simplification, show how it is encoded as an Integer Linear Program, and elaborate on a method to select an appropriate cost budget.

Optimal Folding Simplification. Let G be a GSPN, F be a set of folding instantiations, and ν the respective precedence function. The cost of every folding instantiation $f_i \in F$ is denoted c_i, calculated as described in Section 4. Further, P^3-Folder works with a real-valued budget, $B \in \mathbb{R}^+$, which corresponds to the cumulative error (sum of all costs) that is incurred by the foldings with respect to some performance query $q(Y)$, e.g., the total sojourn time. Last, the utility of every folding instantiation $f_i \in F$, denoted by u_i, is defined as the difference in the number of transitions before and after folding.

The *Optimal Folding Simplification* (OFS) problem involves finding a sequence of folding instantiations of F that respects ν, such that the utility is maximised (the GSPN size is minimised) and the total cost of these foldings does not exceed B.

ILP Encoding. OFS is a tree-knapsack problem, a generalised 0-1 knapsack problem, that is known to be $\mathcal{NP}$-complete. In this problem all items are subjected to a partial ordering represented by a rooted tree [24]. In our case, this partial ordering is induced by the precedence function defined over the folding instantiations.

In what follows, we show a simple reduction from the tree-knapsack problem to an Integer Linear Program (ILP). The ILP problem is well-studied and many tools exist for its solution. We instantiate the ILP as follows. Let x_i be a decision variable that receives 1 if the folding instantiation f_i is applied to G, and 0 otherwise. Then, the ILP representation of the OFS problem is:

$$\underset{x_i}{\text{maximize}} \sum_{i=1}^{n} u_i x_i$$

$$\text{subject to: } \sum_{i=1}^{n} c_i x_i \leq B \quad \wedge \quad \forall\, i,j \in \{1,\ldots,n\}, f_j \in \nu(f_i) : x_i \geq x_j.$$

Here, the score function ensures that total utility is maximized, while the constraints ensure that folding errors do not exceed budget B and that the precedence ν is respected.

Budget Selection. The only input of the OFS problem that is not based on the originating model, G, is the budget B. The budget can be interpreted as the amount of trust in G: B should be small if trust is high, and vice versa.

When applying P^3-Folder to a model G that was constructed based on some event log L, the budget can be set in the spirit of model selection techniques that are often used in machine-learning [25]. Specifically, one may elicit the 'best' budget for a given log via K-fold cross-validation [25, Ch. 7]: The event log is partitioned into K parts, and the budget is determined based on random $\frac{K-1}{K}$ parts that are treated as training logs, and tested on the remaining part. All budgets between 0 (no folding) and $\sum_{i=1}^{n} c_i$ (unlimited folding) are considered and the budget that yields the most accurate answer to the performance query $q(Y)$ under a certain criteria (e.g., sampled root-mean squared error) is selected for the OFS problem.

6 Evaluation

We evaluated P^3-Folder with a real-world case from the healthcare domain. P^3-Folder[3] is implemented in the Python programming language, and uses Gurobi [37], for solving the ILP. The input is a process model (GSPN), and an event log; the method produces a folded GSPN model. Our results indicate that our simplification technique helps to avoid over-fitting of GSPN. P^3-Folder yields up-to a 15% improvement in accuracy when predicting the total sojourn times, with respect to a GSPN discovered from log data using state-of-the-art mining algorithms.

6.1 Datasets and Setup

Our experiments were based on five months (April-August, 2014) of real-world operational data stemming from the treatment process of a large outpatient hospital in the United States. The hospital treats approximately 1000 patients a day, with patients arriving and leaving on the same day. The average length-of-stay per visit is 4.4 hours (standard deviation of 2 hours) with the highest number of patients arriving between 8:00 and 11:00 in the morning. The dataset includes the following attributes: case identifier, activity start time, activity end time, and resource performing the activity.

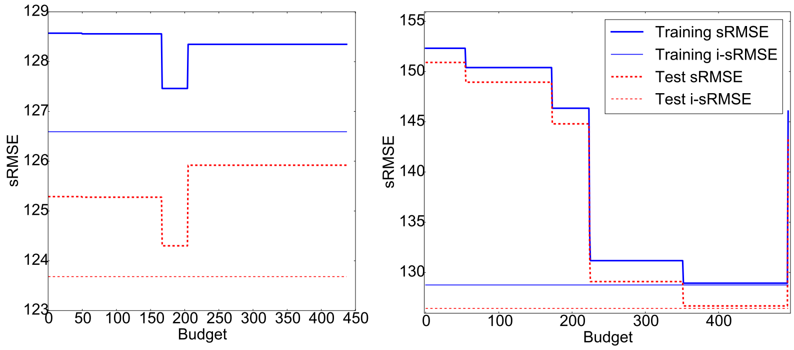
We selected April as our training set for discovering a GSPN and enriching it with data, as well as for the error budget selection (outlined in Section 5). The other four months were used as separate test sets, to validate the results.

To discover an initial Petri net, we applied the Inductive Miner [14] on the training set, with resources being treated as activities and a 20% noise threshold (see Fig. 1b). We enriched the model based on the training set using the techniques described in [12], thus turning it into the initial GSPN.

As the performance query $q(Y)$, we selected the determination of total sojour times. To estimate $q(Y)$ for a given GSPN, we implemented a GSPN-to-queueing networks transformation and used the queueing network analyzer [17].

We focused on three evaluation aspects: (1) We explored the impact of the error budget on the accuracy of the resulting models. To this end, we varied the budget between 0 (no folding) and $\sum_{i=1}^{n} c_i$ (unlimited folding). (2) We studied the sensitivity of the approach to patient volumes, i.e., exploring the improvement in prediction accuracy caused by foldings as a function of time-of-day. We varied the time periods for which the original GSPN was obtained and then selected the best budget with respect to the

[3] `https://github.com/ArikSenderovich/P3Folding/`



(a) All-day. (b) 9:00-10:00 in the morning.

Fig. 4: sRMSE in relation to the error budget used for folding.

training set by cross-validation (Section 5). (3) We considered the interplay of methods to over-fitting in the control-flow dimension (i.e., noise filtering in the initial GSPN discovery) and our approach. We altered the noise filtering threshold in the Inductive Miner, and estimated the prediction accuracy of an unfolded model. We compared the obtained results to those achieved with a folded model.

To quantify the accuracy of models, we used the sample root-mean squared error (sRMSE), which is a standard statistical accuracy measure, defined as follows. Let $\{Y_k\}_{k=1}^{K}$ be the sample of K total sojourn times as observed in the log (training or test). Then, the sRMSE is defined as:

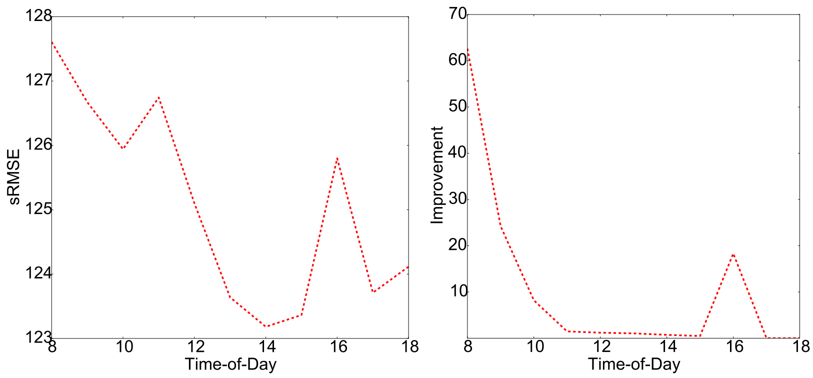
$$sRMSE = \sqrt{\frac{1}{K}\sum_{i=1}^{K}[\hat{q}(Y) - Y_k]^2}.$$

As a baseline method, we used the historical average, which is an unbiased estimator. For the total sojourn time query $q(Y)$, it is the standard deviation of the length of stay, 120 minutes for the entire five months. In sum, controlled variables in our experiments were the budget, the time-of-day, and the noise filtering threshold of the Inductive Miner, while the sRMSE is the response variable.

6.2 Results

First, even though the tree-knapsack problem is known to be $\mathcal{NP}$-complete [24], modern ILP solvers enable efficient reasoning on the OFS problem. Specifically, the run-time of P^3-Folder when considering the entire days of the training data and all budget configurations, turned out to be 152 seconds. This run-time is in the same range as the model discovery, which demonstrates feasibility of the approach.

Next, we turn to the evaluation of the accuracy improvement achieved by P^3-Folder. Fig. 4 shows the sRMSE as a function of the error budget for two time frames, namely all-day and 9:00-10:00 in the morning. Here, the solid blue line corresponds to the training data (April) and the dashed red line to one of the test datasets (May). We demonstrate a single test month, since for fixed time-of-day intervals we did not observe a difference



(a) Absolute sRMSE.

(b) Improvement over original model.

Fig. 5: sRMSE in relation to time-of-day.

in sRMSE shape or value for all four months used as test datasets. The two additional flat lines correspond to the irreducible sRMSE (i-sRMSE) for the training and test sets, respectively. The irreducible error represents a bound for the prediction as it is essentially the noise, the variance of the total sojourn time in the data. Consequently, one cannot improve the sRMSE beyond the i-sRMSE without adding additional predictive features to the model (e.g. number of patients in the system or patient attributes).

We observe that the shape of sRMSE as a function of budget differs for the different time frames. For the all-day scenario, we observe that while low-budget folding improves the sRMSE, high budget folding causes the accuracy to deteriorate. On the other hand, for the busy period of 9:00-10:00, we notice a monotone improvement in the sRMSE as the budget grows and more folding is allowed (with 15% improvement for the maximal budget). Furthermore, for the busy period, our method is able to approach the irreducible error. Lastly, we see that the model trained on the April data has a higher accuracy for the May data, which indicates that P^3-Folder does not suffer from over-fitting the log.

Further, we select the error budget by cross-validation using the training dataset and explore the sensitivity to patient volumes. Fig. 5a depicts the sRMSE as a function of the time-of-day. Fig. 5b shows the absolute improvement in sRMSE, i.e., the difference (in $\%$) between the sRMSE of the original and the folded model. The sRMSE changes over the day. Specifically, Fig. 5b illustrates that our technique is most effective during the morning hours, where the load is highest. This can be the result of our queueing approximation technique [17], as it has accuracy guarantees for heavy-traffic periods.

Finally, we explore the impact of noise filtering in the initial discovery (balancing over-fitting and under-fitting in control-flow) on our method. We alter the noise threshold for the Inductive Miner between 15% and 40% and compute the sRMSE of its unfolded prediction for the 9:00-10:00 interval, and compare the result to the sRMSE of the model obtained by P^3-Folder when folding the 20% noise model. Fig. 6 illustrates that the sRMSE for the unfolded model improves and deteriorates, while the sRMSE of the model obtained by P^3-Folder (Best Model) remains constant. Hence, P^3-Folder finds the optimal level of generalisation for answering the respective performance query.

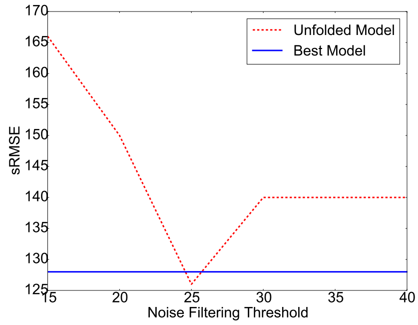


Fig. 6: sRMSE in relation to noise filtering threshold in initial model discovery.

7 Related Work

Previous work on business process simplification (synonymous to abstraction), considers manual ad-hoc rules that simplify the model while preserving similarity to the originating model [26]. Works on automated model simplification proposed to aggregate and eliminate components according to user-defined rules [28]. These rules were concerned mainly with visualization, and preserving behavioural relations between the various components. In [29], process abstraction is consistent, automatic, and preserves behavioural similarities, while in [36] the authors rely on Petri net unfolding into branching processes to balance behavioural over-fitting and under-fitting of a discovered model. However, none of these works considered performance preserving simplifications. Another related approach is to filter the data, prior to applying automated discovery, therefore creating simple models based on partial set of the data [14].

Existing performance-oriented model simplification techniques approximate typical process patterns (e.g., sequence, choice) via queueing theory, and guaranteed certain notion of equivalence between the original model and the resulting simplifications [30,31]. However, these techniques did not propose a method of locating typical patterns, and thus were not automated. Moreover, these works did not suggest how to order the simplifying operations, and some of the proposed performance bounds are not well-grounded [31].

Manual simplification of GSPN models has been considered before. In [32], GSPNs are simplified by using ad-hoc rules, not providing any error bounds. A simplification technique that provides bounds for specific performance measures between the original model and the resulting simple model includes decomposition and aggregation of the GSPN [33,34]. The first step (decomposition) refers to partitioning the GSPN into subnets, such that the subnets are weakly dependent. Every subnet can then be efficiently analysed without unfolding the underlying CTMC [35,31]. The second step (aggregation) aggregates the subnet according to performance-preserving rules.

Our approach takes up the ideas of model aggregation based on folding steps [8]. However, the steps in [8] incorporate ad-hoc assumptions and violating them may yield an unbounded estimation error with respect to the original model. In this work, we formulate an optimization problem aiming at a maximal number of folding instantiations, subject to guarantees regarding an error budget. This enables us to balance performance fitness and generalization of the resulting model in a principled manner.

8 Conclusion

In this work, we presented P^3-Folder as a novel technique for automated simplification of models that aim at improving performance analysis of business processes. Specifically, we proposed foldings of GSPNs and showed how to find an optimal sequence of applying them to obtain a minimal model under a given error budget for the performance estimates. This results in a model that generalises in the performance dimension, while preserving the process perspective of the original model. The evaluation of our technique showed a significant increase in the model's predictive power, with respect to the unfolded model that was discovered from a real-world event log. The proposed technique can be viewed as regularization method for process models, in analogy to pruning and other model selection methods in machine learning.

In future work, we aim at integrating behavioural fitness and performance fitness. Specifically, optimal simplification can be modified to include both the control-flow and time perspective. We further aim at testing the accuracy improvements achieved by our technique on other queries, such as outcome prediction and resource utilisation.

Acknowledgments. This work was partially supported by the German Research Foundation (DFG), grant WE 4891/1-1.

References

1. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
2. Rogge-Solti, A., Weske, M.: Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays. In: ICSOC. LNCS 8274, Springer (2013) 389–403
3. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining - predicting delays in service processes. In CAiSE. LNCS 8484, Springer (2014) 42–57
4. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
5. van der Aalst, W.M.P., Rubin, V., Verbeek, H., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Software & Systems Modeling **9**(1) (2010) 87–111
6. Buijs, J.C., Van Dongen, B.F., van der Aalst, W.M.: On the role of fitness, precision, generalization and simplicity in process discovery. In: OTM 2012. Springer (2012) 305–322
7. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Avoiding over-fitting in ILP-based process discovery. In BPM. LNCS 9253, Springer (2015) 163–171
8. Senderovich, A., Rogge-Solti, A., Gal, A., Mendling, J., Mandelbaum, A., Kadish, S., Bunnell, C.A.: Data-driven performance analysis of scheduled processes. In BPM. LNCS 9253, Springer (2015) 35–52
9. van der Aalst, W.M.P., Schonenberg, M., Song, M.: Time prediction based on process mining. Information Systems **36**(2) (2011) 450–475
10. de Leoni, M., van der Aalst, W.M., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Information Systems **56** (2016) 235–257
11. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: BPM. Springer (2015) 297–313

12. Rogge-Solti, A., van der Aalst, W.M., Weske, M.: Discovering Stochastic Petri Nets with arbitrary delay distributions from event logs. In: BPM Workshops, Springer (2013) 15–27
13. Rozinat, A., Mans, R., Song, M., van der Aalst, W.M.P.: Discovering simulation models. Information Systems **34**(3) (2009) 305–327
14. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM Workshops, Springer (2014) 66–78
15. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with generalized stochastic Petri nets. John Wiley & Sons, Inc. (1994)
16. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains - Modeling and Performance Evaluation with Computer Science Applications. Wiley (2006)
17. Whitt, W.: The queueing network analyzer. Bell Sys. Tech. Journal **62**(9) (1983) 2779–2815
18. van der Aalst, W.M., Ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and parallel databases **14**(1) (2003) 5–51
19. Burke, P.J.: The output of a queuing system. Operations research **4**(6) (1956) 699–704
20. Hall, R.W.: Queueing methods for services and manufacturing. (1990)
21. Balsamo, S., Marin, A.: Composition of product-form Generalized Stochastic Petri Nets: a modular approach. In: Proc. ESM. (2009) 26–28
22. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data and Knowledge Engineering (DKE) **68**(9) (2009) 793–818
23. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. In: WS-FM. LNCS 6551, Springer (2010) 25–41
24. Shaw, D.X., Cho, G.: The critical-item, upper bounds, and a branch-and-bound algorithm for the tree knapsack problem. Networks **31**(4) (1998) 205–216
25. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics. Springer New York Inc., New York, NY, USA (2001)
26. Smirnov, S., Reijers, H.A., Weske, M., Nugteren, T.: Business process model abstraction: a definition, catalog, and survey. Distributed and Parallel Databases **30**(1) (2012) 63–99
27. Resnick, S.I.: Adventures in stochastic processes. Springer Science & Business Media, New York, NY, USA (2013)
28. Günther, C.W., van der Aalst, W.M.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: Business Process Management. Springer (2007) 328–343
29. Mafazi, S., Grossmann, G., Mayer, W., Schrefl, M., Stumptner, M.: Consistent abstraction of business processes based on constraints. Journal on Data Semantics **4**(1) (2014) 59–78
30. Zerguini, L.: On the estimation of the response time of the business process. In: 17th UK Performance Engineering Workshop, University of Leeds, Citeseer (2001)
31. Zerguini, L., van Hee, K.M.: A new reduction method for the analysis of large workflow models. In: Promise. (2002) 188–201
32. Balbo, G., Bruell, S.C., Ghanta, S.: Combining queueing networks and generalized stochastic Petri nets for the solution of complex models of system behavior. Computers, IEEE Transactions on **37**(10) (1988) 1251–1268
33. Ciardo, G., Trivedi, K.S.: A decomposition approach for stochastic Petri net models. In: Petri Nets and Performance Models, IEEE (1991) 74–83
34. Woodside, C.M., Li, Y.: Performance petri net analysis of communications protocol software by delay-equivalent aggregation. In: Petri Nets and Performance Models, IEEE (1991) 64–73
35. Freiheit, J., Billington, J.: New developments in closed-form computation for GSPN aggregation. In: ICFEM, Springer (2003) 471–490
36. Fahland, D. and Van Der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Information Systems **38**(4) (2013) 585–605
37. Gurobi Optimization, Inc.: Gurobi Optimizer Reference Manual http://www.gurobi.com (2015)