Probabilistic Programming for Trace Generation (and Beyond)

Martin Kuhn[1], Joscha Grüger[2], Christoph Matheja[3,4] and Andrey Rivkin[4]

[1]German Research Center for Artificial Intelligence (DFKI), SDS Branch Trier, Trier, Germany
[2]University of Trier, Trier, Germany
[3]University of Oldenburg, Oldenburg, Germany
[4]Technical University of Denmark, Kgs. Lyngby, Denmark

Abstract

Probabilistic programming is a programming paradigm that enables the creation of probabilistic models, their simulation through execution, and their analysis through various inference engines. Recently, it has been applied in a framework for simulating Data Petri Nets (DPNs) – a class of Petri nets widely used in the business process community for modeling and reasoning about data-aware processes. In this paper, we summarize recent advances in this area and outline potential directions for future research. We hope this will highlight the potential of the synergy between probabilistic programming and process science, and inspire further exploration at the intersection of these fields.

Keywords

Probabilistic Programming, Trace Generation, Data Petri Nets

1. Introduction

Process simulation is a powerful concept that has numerous applications in Business Process Management and Process Mining [1, 2]. Process simulation models can be discovered from event logs [3, 4, 5, 6] and used for tasks such as process redesign [7], what-if analysis and process performance improvement [2], synthetic log generation [8], and augmented process execution [9].

The majority of simulation techniques rely on discrete event simulation (DES) [10, 2] or hybrid approaches like in [11]. In DES, a frequentist perspective is typically adopted, treating all "unknowns" (i.e., probabilities) as fixed frequencies a priori known by the domain expert.

In our recent paper [12], we explored an alternative simulation approach grounded in Bayesian statistics.[1] Specifically, we proposed using probabilistic programming (PP) [14] to represent business processes as statistical models, and carry out (stochastic) simulation using a PP inference engine, which samples different process simulation parameters from probability distributions provided by a randomized scheduler that must be supplied together with the input process model. The framework proposed in [12] was implemented in a prototype [15], which has so far been applied primarily to trace generation scenarios.

In this paper, we provide a brief overview of our simulation approach and outline promising directions for supporting more advanced simulation scenarios and analysis tasks.

2. The Approach

Simulatable process models. Our approach takes as input imperative process models enriched with probability distributions over their non-deterministic decision points. These distributions offer a

ATAED'25, International Workshop on Algorithms & Theories for the Analysis of Event Data, 2025
martin.kuhn@dfki.de (M. Kuhn); grueger@uni-trier.de (J. Grüger); christoph.matheja@oul.de (C. Matheja); ariv@dtu.dk (A. Rivkin)
0000-0002-3242-1251 (M. Kuhn); 0000-0001-7538-1248 (J. Grüger); 0000-0001-9151-0441 (C. Matheja); 0000-0001-8425-2309 (A. Rivkin)

[1]The advantages of Bayesian approaches in comparison to DES are discussed in [13].

structured way to manage non-determinism [16] and allow the incorporation of domain knowledge (e.g., execution rates), enabling more realistic and flexible simulation scenarios.

There are a few ways in which such models can be obtained. One option is to directly discover stochastic models from event logs [17, 18, 19]. Alternatively, one can apply separate techniques to discover the control-flow model and the associated simulation parameters, as in [5]. Another approach involves defining randomized schedulers handling non-deterministic choices by assigning probability distributions to respective decision points [12].[2] We elaborate on this last approach below.

It is important to note that schedulers offer greater flexibility by assigning full probability distributions (rather than fixed stochastic weights) to decision points. These can also be derived from the aforementioned discovery methods. However, manually defining such schedulers is challenging, as it requires both in-depth knowledge of the process and expertise in probabilistic modeling to properly map business requirements to meaningful probability distributions.

In [12], we proposed a scheduler-based simulation approach for data Petri nets (DPNs for short) [20, 21, 22]. A DPN is a labeled place-transition net (in [12] it is considered without silent transitions) extended with guards defined over a finite set of (typed) process variables. Each guard consists of two boolean expressions: a pre-condition and a post-condition. Each DPN state is represented as a pair (M, α), where M is a marking (token distribution) and α is a process variable valuation. A transition t is enabled in a state (M, α) if *(1)* M contains enough tokens for t to fire (according to the net's flow relation), and *(2)* the pre-condition of t is satisfied under variable valuation α. Upon firing, a transition consumes/produces tokens (as per the standard execution semantics of P/T nets [23]) and updates variables based on its post-condition.

It is easy to see that DPNs can induce infinitely many distinct states. Each such state is the result of performing 1. non-deterministic choices between multiple enabled transitions, and 2. non-deterministic selection of variable valuations satisfying the post-condition (e.g., a post-condition $x' > 0$ defined over a real-valued variable x is satisfied by any positive real number) of the chosen transition. To resolve both sources of non-determinism, in [12] we defined randomized schedulers. For each state (M, α), the scheduler specifies a probability distribution over net's transitions and, for each variable modified by a transition, a probability distribution over its domain. Transition selection is resolved by sampling a transition according to the provided transition distributions. If some transitions are not enabled, their probability is redistributed uniformly among the enabled ones. Once a transition is selected, values for the variables modified by its post-condition are picked using the corresponding distributions assigned to each such variable.

Coupling a DPN N with a randomized scheduler Sc has two outcomes. First, we can define a probabilistic execution semantics for DPNs, enabling the computation of the probability of a run (i.e., a sequence of transitions and variable valuations) reaching a given symbolic goal G (e.g., a marking with two tokens assigned to place p_5 and process variables $x, y \in [-2.3, 100]$). This is conceptually similar to computing reachability probabilities in infinite-state Markov chains [24, 25]. We can thus say that a run reaches a goal if its probability is non-zero. Second, given probability distributions for all decision points of our process model, we can attempt to sample runs with non-zero probabilities, and extract a process trace from each successfully sampled run. We describe below how this can be achieved using the PP machinery.

Encoding into a PP language and simulation. For our approach, in [12] we introduced a probabilistic guarded command language (called PPL) incorporating key features commonly found in probabilistic programming (PP) languages [26], and its non-probabilistic core is closely aligned with Dijkstra's Guarded Command Language (GCL) [27]. The main constructs of PPL are as follows.

(1) Probabilistic assignment statement $x := D$ assigns probability distribution D to program variable x with which values will be sampled and assigned to x. For example, $x := \mathsf{uniform}(1, 3)$ defines that x will be randomly assigned a value between 1 and 3 with probability $1/3$ each.

(2) $\mathsf{observe}\ B$ is the conditioning command that incorporates observed evidence (in the shape of predicate B) into the probabilistic program simulation. If the predicate B holds, then the simulation

[2] The mechanism of schedulers is commonly used in discrete-event simulation [16].

continues. If B fails, then it means that the simulation encountered an unrealistic result that should be discarded. This blocks the ongoing simulation and, for example, can continue with obtaining another sample for variables involved in B by restarting the whole simulation process.

(3) The $\log Y$ command writes message Y into the program's append-only log.

(4) Probabilistic guarded command $B \xrightarrow{E} C$ executes command C with probability defined in the expression E if the guard defined by the predicate B holds. Such commands can be used either in the unbounded loops or conditional statements.[3] When used in unbounded loops, potentially multiple probabilistic guarded commands will be executed until none of their guards hold, which in turn leads to the loop termination. If a loop consists of multiple $B_i \xrightarrow{E_i} C_i$ commands and at least two distinct B_k and B_l hold, then their respective commands will be executed randomly by taking into account the probabilities in E_k and E_l. After a command is executed, the loop repeats. When used in conditional statements, only one matching guarded command is executed, after which the statement terminates.

Execution of a PPL program corresponds to stochastic simulation: values are sampled from defined distributions and propagated through the program. Moreover, modern PP platforms provide inference techniques to compute or approximate the probability distribution modeled by a program

The encoding of a DPN N and its scheduler Sc into a PPL program P is rather straightforward and systematically described in [12]. Here, we provide a high-level overview of our approach which should also illustrate its potential applicability to other process modeling formalisms.

We start by creating a PPL variable for each DPN place and variable, enabling a 1-to-1 mapping between DPN and PPL program states. These variables are initialized based on the initial state of N. Each DPN transition t is encoded as guarded commands of the form $\neg G \wedge \mathsf{enabled}(t) \xrightarrow{Sc(t)} \mathsf{fire}(t)$, where *(1)* G is the symbolic goal expression discussed above, *(2)* the $\mathsf{enabled}(t)$ expression performs the enabledness check of t by also taking into account its pre-condition, *(3)* $Sc(t)$ provides the transition selection probability (only if t is enabled), *(4)* $\mathsf{fire}(t)$ is a subroutine that simulates the effect of firing t. The $\mathsf{fire}(t)$ subroutine performs the following steps: *(1)* update the DPN marking using the standard marking equation; *(2)* sample temporary values for variables updated by t's post-condition (stored in an auxiliary set of variables), according to the scheduler *(3)* enforce the satisfaction of the post-condition on the sampled values by executing the observe statement on the post-condition of t; *(4)* log the information about the fired transition using log; *(5)* commit the sampled values to the DPN variables.

PPL programs can be then executed in any PP environment. Our prototype implementation [15] currently targets WebPPL [28] – a PP language that comes with a versatile inference engine used for simulating probabilistic programs. To configure a simulation scenario in our tool for a given DPN, one has to define the scheduler (by providing probability distributions for transitions and variables modified in their post-conditions), and provide simulation parameters such as the number of traces to generate and maximum trace length. We reported on our preliminary evaluation in [15], showing highly promising performance results.

Lastly, it is important to mention that with our approach we obtain a useful guarantee stipulating that the probability of each generated trace matches the probability of the corresponding DPN run induced by the selected scheduler.

3. Applications and Future Directions

Our preliminary investigation primarily aimed at establishing a foundational connection between a class of data-aware Petri nets and a probabilistic programming (PP) language. The feasibility of this integration was confirmed, and further experiments demonstrated that our PP-based simulation approach is capable of generating a substantial number of traces in reasonable time.

Importantly, given that the resulting programs can be both simulated and analyzed using statistical inference engines, our approach offers applications that go beyond mere trace generation. We highlight

[3] Due to space limitations, we refer to [12] for a full description of the PPL syntax.

two particularly promising use cases:

1: Rare event simulation and beyond. The use of inference engines enables the computation of conditional probabilities for rare or domain-specific events, expressed as boolean formulas, similarly to the way we specify G. An example of such formula φ is $x \neq 10.5 \wedge 4 \geq \#t_1 \leq 6$, specifying a condition for reaching a final marking according to which x differs from 10.5 and transition t_1 fires between 4 and 6 times (here $\#t_1$ is a new variable that must be additionally introduced into the program). Given the aforementioned goal of obtaining simulations in which the final marking is reached, we can add to our program the statement ovbserve φ allowing us to restrict the simulation to those executions satisfying φ and obtain a conditional probability distribution over such executions (which correspond to process traces) only. This paradigm allows us to simulate and analyze low-probability or otherwise constrained behavior without modifying the underlying process model (and thus not affecting its PPL encoding). This makes it useful for tasks such as risk analysis or behavioral auditing.

2: What-if analysis. To support hypothesis-driven exploration of new behaviors in the given DPN, one can perform what-if analysis by modifying the scheduler and/or adding new observe φ statements, similarly to the rare-event simulation setup. Beyond that, this approach also does not require altering the DPN structure or its corresponding PPL encoding. However, crafting a scheduler that faithfully models a behavioral hypothesis provided for a process can be a complex task, as mentioned previously. This demands both deep domain expertise and strong understanding of probabilistic modeling to ensure that modifications are meaningful and semantically coherent.

While these applications are within reach of the current prototype [15], they expose limitations in the broader usability and extensibility of the approach.

- One of the limitations that the current approach has is the lack of a dedicated, well-defined and user-friendly language for specifying goal predicates or formulas used in the ovbserve φ statements. This clearly limits the adoption of our approach in practice as introducing new goals or conditioning statements requires direct modification of the generated PPL code. Identifying or designing such a language, along with its systematic translation to PPL, is a promising direction for future work and an open challenge in PP-based process simulation.
- Another clear limitation of our approach is the lack of structured log generation capabilities. Although multiple traces can be generated via repeated simulation, the resulting trace collections lack statistical coherence and realism when combined into a process log. For instance, generating 100 traces of length 15 may yield a diverse set which is representative of the process, while generating 100 traces of length 32 (for the same process) may result in overfitting or skewed execution paths due to, e.g., marginal variations in sampled values obtained for one process variable. This issue underscores the need for advanced mechanisms that can retain and leverage simulation history and most surely should be addressed by extending the PPL language with memory-aware/stateful commands that can be used to guide trace generation based on previous outcomes. Such an extension would move us closer to the generation of realistic, statistically meaningful process logs.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: paraphrase and reword the text. After using this tool/service, the author(s) reviewed and edited the content as needed, and take(s) full responsibility for the publication's content.

References

[1] J. Ryan, C. Heavey, Process modeling for simulation, Computers in Industry 57 (2006) 437–450.

[2] W. M. P. van der Aalst, Process mining and simulation: a match made in heaven!, in: SummerSim, ACM, 2018, pp. 4:1–4:12.

[3] A. Rozinat, R. S. Mans, M. Song, W. M. P. van der Aalst, Discovering simulation models, Inf. Syst. 34 (2009) 305–327.

[4] N. Martin, B. Depaire, A. Caris, The use of process mining in business process simulation model construction - structuring the field, Bus. Inf. Syst. Eng. 58 (2016) 73–87.
[5] M. Camargo, M. Dumas, O. González, Automated discovery of business process simulation models from event logs, Decis. Support Syst. 134 (2020) 113284.
[6] O. López-Pintado, S. Murashko, M. Dumas, Discovery and simulation of data-aware business processes, in: Proc. of ICPM 2024, IEEE, 2024, pp. 105–112.
[7] L. Maruster, N. R. T. P. van Beest, Redesigning business processes: a methodology based on simulation and process mining techniques, Knowl. Inf. Syst. 21 (2009) 267–297.
[8] A. Burattin, B. Re, L. Rossi, F. Tiezzi, A Purpose-Guided Log Generation Framework, in: C. Di Ciccio, R. Dijkman, A. del Río Ortega, S. Rinderle-Ma (Eds.), Business Process Management, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 181–198.
[9] D. Chapela-Campa, M. Dumas, From process mining to augmented process execution, Softw. Syst. Model. 22 (2023) 1977–1986.
[10] L. Pufahl, T. Y. Wong, M. Weske, Design of an extensible BPMN process simulator, in: E. Teniente, M. Weidlich (Eds.), Proc. of BPM Workshops 2017, volume 308 of *LNBIP*, Springer, 2017, pp. 782–795.
[11] M. Pourbafrani, W. M. P. van der Aalst, Hybrid business process simulation: Updating detailed process simulation models using high-level simulations, in: R. S. S. Guizzardi, J. Ralyté, X. Franch (Eds.), Proc. of RCIS 2022, volume 446 of *LNBIP*, Springer, 2022, pp. 177–194.
[12] M. Kuhn, J. Grüger, C. Matheja, A. Rivkin, Data petri nets meet probabilistic programming, in: A. Marrella, M. Resinas, M. Jans, M. Rosemann (Eds.), Proc. of BPM 2024, volume 14940 of *LNCS*, Springer, 2024.
[13] S. E. Chick, Bayesian ideas and discrete event simulation: why, what and how, in: L. F. Perrone, B. Lawson, J. Liu, F. P. Wieland (Eds.), Proc. of WSC, IEEE Computer Society, 2006, pp. 96–105.
[14] A. D. Gordon, T. A. Henzinger, A. V. Nori, S. K. Rajamani, Probabilistic programming, in: FOSE, ACM, 2014, pp. 167–181.
[15] M. Kuhn, J. Grüger, C. Matheja, A. Rivkin, Logppl: A tool for probabilistic process mining, in: J. D. Weerdt, G. Meroni, H. van der Aa, K. Winter (Eds.), Proc. of Doctoral Consortium and Demo Track, ICPM, volume 3783, CEUR-WS.org, 2024.
[16] A. M. Law, Simulation Modeling & Analysis, McGraw-Hill, NY, USA, 2015.
[17] S. J. J. Leemans, T. Li, M. Montali, A. Polyvyanyy, Stochastic process discovery: Can it be done optimally?, in: G. Guizzardi, F. M. Santoro, H. Mouratidis, P. Soffer (Eds.), Proc. of CAiSE, volume 14663 of *LNCS*, Springer, 2024, pp. 36–52.
[18] S. J. J. Leemans, T. Li, J. N. van Detten, Ebi - a stochastic process mining framework, in: J. D. Weerdt, G. Meroni, H. van der Aa, K. Winter (Eds.), Proc. of Doctoral Consortium and Demo Track, ICPM, volume 3783 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2024.
[19] F. Mannhardt, S. J. J. Leemans, C. T. Schwanen, M. de Leoni, Modelling data-aware stochastic processes - discovery and conformance checking, in: L. Gomes, R. Lorenz (Eds.), Proc. of PETRI NETS 2023, LNCS, Springer, 2023.
[20] F. Mannhardt, M. de Leoni, H. A. Reijers, W. M. P. van der Aalst, Balanced multi-perspective checking of process conformance, Computing 98 (2016).
[21] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, Cocomot: Conformance checking of multi-perspective processes via SMT, in: A. Polyvyanyy, M. T. Wynn, A. V. Looy, M. Reichert (Eds.), Proc. of BPM 2021, volume 12875 of *LNCS*, Springer, 2021, pp. 217–234.
[22] P. Felli, A. Gianola, M. Montali, A. Rivkin, S. Winkler, Conformance checking with uncertainty via SMT, in: Proc. of BPM 2022, LNCS, Springer, 2022.
[23] T. Murata, Petri nets: Properties, analysis and applications, Proc. IEEE 77 (1989) 541–580.
[24] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st ed., John Wiley & Sons, Inc., USA, 1994.
[25] C. Baier, J. Katoen, Principles of model checking, MIT Press, 2008.
[26] J. Katoen, F. Gretz, N. Jansen, B. L. Kaminski, F. Olmedo, Understanding probabilistic programs, in: R. Meyer, A. Platzer, H. Wehrheim (Eds.), Proc. of Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, volume 9360 of *LNCS*, Springer, 2015, pp. 15–32.

[27] E. W. Dijkstra, A Discipline of Programming, Prentice-Hall, 1976.
[28] N. D. Goodman, A. Stuhlmüller, The Design and Implementation of Probabilistic Programming Languages, 2014.