
Converting Population-Based Simulators to Probabilistic Programs

Paper ID:

Abstract

Probabilistic programming provides a way to perform statistical inference over simulations of events in a programmatic way. However, with the myriad of probabilistic programming systems (PPSs) and the complexities involved in running each individual system, such as specifying a model in the given language and \ or running inference, which means it can be difficult for users to take advantage of the system. In addition to this, even though there exists a myriad of simulators across many scientific domains and within industry, none can be used within existing PPSs without having to re-implement the simulator in the given PPS, something that is not typically feasible, nor practical. However, recent work by [2] demonstrated a pathway for how to turn a particular type of simulator, written in a set of 9 languages, into a probabilistic program via the raw random number calls. But, this still meant that a large class of critically important population-based simulators could not be turned into probabilistic programs. We, in this work, extend that framework to encompass population-based simulators, a very large class of simulators that are used extensively across epidemiology modeling, multi-agent modeling, financial modeling and so forth. In addition to this, we enable simulators written in 13 different languages to be easily converted in probabilistic programs, which we algorithmically outline. We demonstrate our PPS on a both a toy population simulator and two important malaria population-based simulators called OpenMalaria and EMOD.

1 Introduction

- Outline the objectives
- State the importance of interpretability in the inference of the simulator
- Our system enables one to understand and interpret the most common paths in a program (simulation).
- In highlighting this a practitioner can then go back to the field and explore that parameter/s in more depth.
- This facilitates understanding and the construction of more detailed models and data collection strategies

[BG: The audience is a probabilistic programming audience - so that is what we will target]

In this paper, we introduce a novel method that allows one to shed light on the inner workings of a large class of population-based stochastic simulators. We achieve this by extending the work of [2] by interpreting such population-based simulators as probabilistic generative models within the framework of universal probabilistic programming (UPP) [12]. To this end, we *hijack* existing simulators by overriding their internal random number generators. Specifically, by replacing the existing low-level random number generator in a simulator with a call to a purpose-built UPP “controller”, which can thus control, track and manipulate the stochasticity of the simulator.

This allows for a variety of tasks to be performed on the hijacked simulator, such as running inference (by conditioning the values of certain draws and manipulating others), uncovering stochastic structure, and automatically producing result summaries, such as establishing the probability of different program paths/traces. By providing a common abstraction framework for different simulators, our approach further allows for easy and direct comparison between related or competing simulators, a characteristic that is valuable in the context of epidemiology simulators [7]. We provide a case study of the above in Section 8.

Our framework already supports application to simulators written in 13 industrial programming frameworks, and is easily extensible. This is crucial as, given the enormous code size and complexity, rewriting epidemiology simulators using a dedicated universal probabilistic programming language, such as Pyro [4], is often infeasible.

In time, we hope our approach will play a critical role in bringing recent advancements in probabilistic programming to bear on the vast array of existing simulators used throughout the sciences, thereby providing wide-ranging impacts across a number of fields.

This paper first gives an overview of existing Malaria simulators (Section 5), and proceed by introducing the necessary background on the *pyprob* framework and the concept of universal probabilistic programming (Section 7). Our approach is then demonstrated and analysed in the context of a Malaria case study (Section 8).

2 Epidemiology Simulators and Probabilistic Programming

This will kind of be like a background section.

Algorithm 1 How to perform posterior inference over a stochastic simulator with

```

1: function MERGE( $A, p, q, r$ ) ▷ Where A - array, p - left, q - middle, r - right
2:   replace raw samples statements
3:    $n_2 = r - q$ 
4:   Create Containerized Environment
5:   for  $i = 1$  to  $n_1$  do
6:      $L[i] = A[p + i - 1]$ 
7:   end for
8:   for  $j = 1$  to  $n_2$  do
9:      $R[i] = A[q + j]$ 
10:  end for
11:   $L[n_1 + 1] = \infty$ 
12:   $R[n_2 + 1] = \infty$ 
13:   $i = 1$ 
14: end function

```

3 Types of Models and Probabilistic Programming Languages

To bridge the gap between the statistics and machine learning community to the probabilistic programming community we provide a useful overview to expose our ideas more concretely.

Probabilistic programming languages, broadly speaking¹, can be classified into three categories, each type of language extends a particular model space, enabling an increasingly complex set models to be represented as programs, but in doing so the notion of posterior and what it means to do inference over the model, at least in a stastically traditional sense, can change. The first type of languages are called first-order languages where the priors are well defined and the *joint* program density constructed via the program directly represent the unnormalised stastical joint density $p(x, y)$

4 Methodology

TODO

¹For a more comprehensive overview see [15]

Simulators encode several years, if not decades, of research and development and so by their very nature are structurally complicated. Since the development happens over several decades, usually without documentation, understanding a code base written with legacy libraries and software is incredibly difficult and an almost impossible task for a non-expert, or expert who has not had access to the previous years of development. Our method of hijacking simulators builds on the work of [2] and extends the framework to encapsulate a more diverse range of simulators, in particular epidemiology simulators. Our framework provides a simple solution and makes it easy to transform arbitrary stochastic simulators into probabilistic programs, regardless of the complexity of the simulator and the language that the simulator is written in². In addition to this, we can understand the structure of programs in simulators for both decoding *black-box structures* and for posterior inference. This enables two things. 1) It enables software developers and researchers to understand complicated code bases. 2) It provides interpretable inference results that provide policy makers with predictions that are truly interpretable, in the sense that the end-user of the inference results understands what physical events led to inference outcome. This is critical in several domains, and is particularly important in the medical domain. We provide examples of both in Section **TO ADD REF**.

5 Simulating Diseases

In-silico simulators have become a crucial tool in evidence-based decision-making within a large number of disciplines, including statistical physics [11], financial modeling [9], weather prediction [6], epidemiology [17] and many others. In many cases, simulation output can augment or even replace real data that may otherwise be costly or even impossible to generate. Recent advances in hardware have enabled simulations to model increasingly complex systems. Epidemiology studies the prevalence and spreading of diseases across populations. Recent advances in hardware have enabled simulations to model the dynamics of infectious diseases, such as Malaria, in ever greater detail.

6 Experiments

6.1 Epidemiology Simulators

Malaria epidemiology is governed by a complex set of drivers, few of which can be understood in isolation [5, 1, 17, 3]. These include within-host dynamics, population-specific traits and even local geography. Comprehensive modeling of all of these components remains challenging, particularly in a region-specific context. Computational epidemiology simulators have to reflect these complexities and are usually stochastic in nature. This can make simulation output highly non-trivial to interpret, particularly when trying to draw desired inferences coupled with observed data [13, 7].

Two the most advanced Malaria simulators, namely EMOD [3] and OpenMalaria [17], have proven to be particularly valuable to policymakers. OpenMalaria is based on microsimulations of *Plasmodium falciparum* in humans and was originally developed to simulate the impacts of malaria vaccines within simple villages or districts. Compared with OpenMalaria, EMOD is able to simulate a variety of additional drivers, including complex geographies complete with migration and a large number of policy interventions. Both EMOD and OpenMalaria are open source and implemented in C++.

[BG: Add linking stuff now to the toy-example where hopefully we can perform inference and]

7 Hijacking Simulators

Probabilistic programming [8, 19, 10] can be used to express probabilistic models and consequently perform automated inference these. Once a probabilistic model has been expressed in a probabilistic programming language, a wide range of inference techniques, such as Markov Chain Monte Carlo (MCMC)[], Variational Inference (VI)[] and Deep Neural Networks (DNN)[], can be used by non-experts in an automated fashion.

Hijacking a simulator describes the process by which a simulator’s random number generators are replaced by calls to external sampling procedures, which are controlled by a probabilistic pro-

²The framework currently supports 9 popular languages, but there is nothing to stop this being extended to any language of interest.

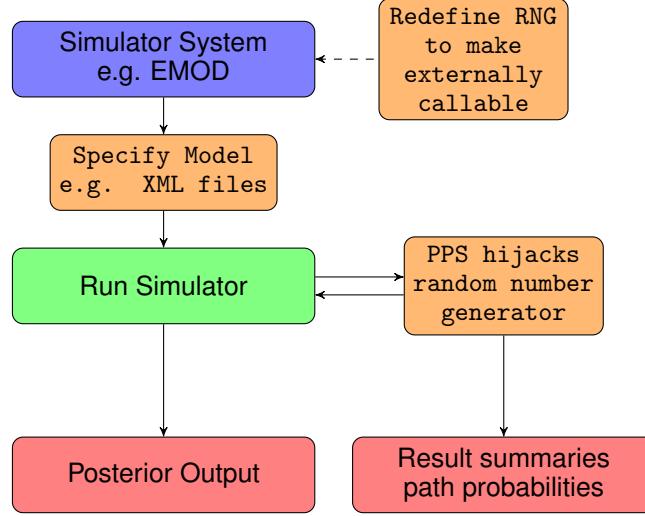


Figure 1: This flow-chart provides an overview of the process of how our system hijacks a generic population-based epidemiology simulator, such as EMOD, and how we modify the simulator to hijack the random number generator (RNG).

gramming system (PPS). In practice, this amounts to performing a small number of surgical incisions into the simulator’s source code in order to replace built-in calls to random number generators. E.g., given a simulator written in C++/Boost[16], a Gaussian distribution object `boost::normal` is replaced with the corresponding `pyprob_cpp` distribution object, namely `pyprob_cpp::distributions::Normal`. Sampling from this distribution is then done by requesting the PPS to send a sample back to the simulator.

The PPS and the simulator exchange xTensor objects <https://xtensor.readthedocs.io/en/latest/> through TCP or ICP using a generic FlatBuffers <http://google.github.io/flatbuffers/> protocol. On the PPS side, sampling is done using the deep learning framework PyTorch [14].

After a variable has been sampled, it is sent to the PPS using `pyprob_cpp::sample`, alongside with its simulator source trace. This allows the PPS to construct sample trace probabilities and other summary statistics (cf. Section 8).

Finally, the entry point to the simulator, i.e. `main` in C++, is replaced by a special `forward` call, in this case `pyprob_cpp::forward`. This allows the PPS to generate rollouts from the simulator remotely.

Population-based simulators In contrast to the setup used by [2], who interface a particle physics simulator that creates a single trace per *forward* call, a population-based simulator creates a trace per population member. This means that e.g. in EMOD, a standard scenario simulation rollout over the span of 3 years with a population of size $n = 5000$ [17] will generate about *four terabytes* of raw trace data. Unlike with [2], this amount of data cannot possibly be kept in RAM, which makes *a posteriori* trace analysis very inefficient. In order to deal with the shortcomings of `pyprob` in a population-based simulator context, we therefore extend the framework to be able to do trace analysis on the fly.

By extending `pyprob` to handle population-based simulators, the PPS can now track all stochastic random variables that are created within the simulator, which then allows us to generate trace plots and path probabilities associated to the execution paths of the program. The corresponding increase of simulator transparency helps reinstate much needed trust between policymakers and evidence-based methods.

8 Case Study: Ifakara (Tanzania)

Ensemble methods are commonly used in statistics in order to combine the predictive power of multiple models [5, 18]. To this end, recent work has attempted to characterise the similarities and differences between two of the most advanced Malaria epidemiology simulators, EMOD [3] and OpenMalaria [17]. Evaluation is usually done by comparing a number of output parameters across a range of hand-crafted standard scenarios reflecting different geographical locations across Africa [18].

In the following, we illustrate how our method introduces a novel introspection paradigm. By extracting trace graphs from population-based simulators, policymakers can ask specific questions about properties of the model trace flow.

To illustrate the above, we present simulation output generated from a scenario resembling local conditions in the town of Ifakara (Tanzania). Both EMOD and OpenMalaria are configured to simulate a single population node of $n = 100$ and assume constant climatic conditions and no migration over the simulation period of 3 years. Please refer to Figure 2 for the seasonal Entomological Inoculation Rate (EIR), a measure of infectivity.

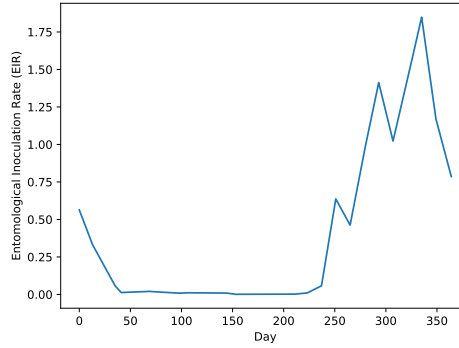


Figure 2: Seasonal Entomological Inoculation Rate (EIR) for the Ifakara scenario. Data is averaged over 30 day periods.

We provide examples of the generated trace plots from the connection between the simulator and the PPS in Figure 3.

We can see from the addressing schemes A_1, \dots, A_N (Tables 2 and 8) what physical events are connected to each other and how outputs in EMOD are generated from a different set of procedures as compared to OpenMalaria. By having access to such diagrams users can internally evaluate and scrutinize the decisions that the simulator is making.

This is important for policymakers, or general non-experts, as it not only details how we arrive at the given outputs, but it provides an understanding of which processes were most crucial in determining those outputs as can be seen from the path probabilities assigned to each of the vertices.

Additionally, by associating nodes in trace graphs representing the same physical processes within different models, the significance of model detail can be evaluated. For example, the full trace presented in Table 8 represent a sampling step associated with within-host dynamics of the Malaria parasite *Falciparum* in OpenMalaria node A1. The same physical process also occurs in EMOD's trace graph at position A7 (see Section 2). Comparisons like these could help developers better control model complexity, and even provide an alternative testing and debugging paradigm.

9 Discussions and Future Work

In this work we have demonstrated a method that enables one to hijack populationbased simulators, extending the work of [2]. We applied our method to two Malaria orientated population-based simulators and generated a variety of trace graphs. Finally, we have shown how our system enables policy makers and non-experts to analyse simulator outputs in a way previously unavailable in the field of epidemiology.

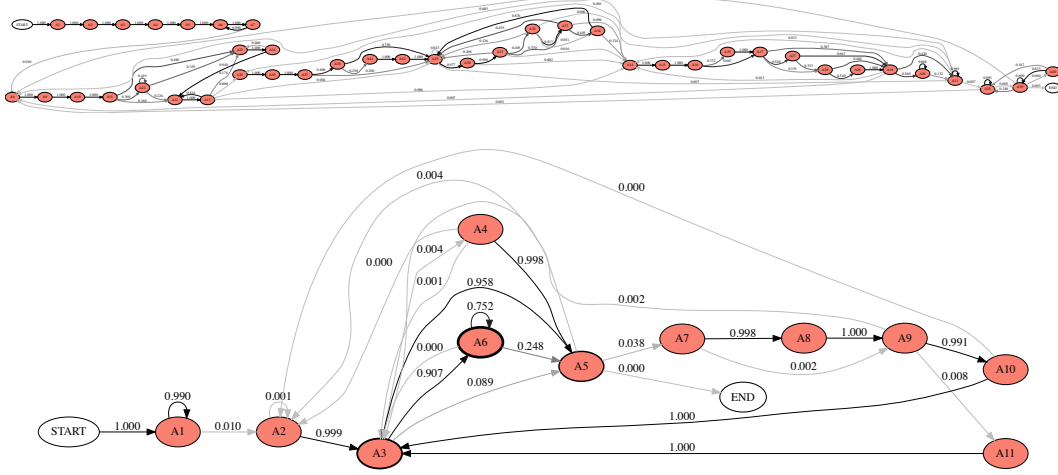


Figure 3: Here we run two equivalent models, compare the corresponding trace paths and corresponding path probabilities taken by the thousands of random variables generated internally within the simulators. **Top:** The specified model run in EMOD. **Bottom:** The specified model in OpenMalaria.

Table 1: An example of an address generated for the model run in the OpenMalaria simulator. We can see that A1 relates to Generating a member of the human population who may or may not be infect with the Malaria disease. We get something similar for EMOD, except this relates to A7 in the EMOD program execution.

Address ID	Full address
A1	[forward()+0x204;OM::Simulator::start(scenXml::Monitoringconst)+0x28a; OM::Population::createInitialHumans()+0x94; OM::Population::newHuman(OM::SimTime)+0x5c; OM::Host::Human::Human(OM::SimTime)+0x12b; OM::WithinHost::WHInterface::reateWithinHostModel(double)+0x99; OM::WithinHost::DescriptiveWithinHostModel::DescriptiveWithinHostModel(double)+0x3a; OM::WithinHost::WHFalciparum::WHFalciparum(double)+0xe6; OM::util::random::gauss(double, double)+0xb4]__Normal
⋮	⋮
A5	[forward()+0x204;OM::Simulator::start(scenXml::Monitoringconst&)+0x468; OM::Population::update1(OM::SimTime)+0xff; OM::Host::Human::update(bool)+0x2bc; OM::Clinical::ClinicalModel::update(OM::Host::Human&,double, bool)+0x96; OM::Host::NeonatalMortality::eventNeonatalMortality()+0x9; OM::util::random::uniform_01()+0xc0]__Uniform
⋮	⋮

To extend our work further we aim to implement additional tools that will facilitate complicated inference procedures that condition on simulator output. We will also evaluate additional scenarios across Africa and South-East Asia to better understand the similarities and differences between EMOD and OpenMalaria.

References

- [1] Beatrice Autino, Alice Noris, Rosario Russo, and Francesco Castelli. Epidemiology of malaria in endemic areas. 4(1).
- [2] Atılım Gunes Baydin, Lukas Heinrich, Wahid Bhimji, Bradley Gram-Hansen, Gilles Louppe, Lei Shao, Kyle Cranmer, Frank Wood, et al. Efficient probabilistic inference in the quest for

Table 2: An interpretation table for each of the address of the overall trace generated from the corresponding OpenMalaria model.

Address ID	Interpretation
A1	Generate a human in the population within host dynamics
A2	Generate another human in the population within host dynamics
A3	The population is updated and a new human, or humans, may get infected
A4, A5	Potential child deaths within the population are simulated
A6	Determines parasite density of an individual infection
A7	Models how the disease is progressing within the infected humans
A8	Models how the disease is progressing within the population
A9	Models how the disease is progressing within the infected humans after the population has been updated
A10	Full clinical update on the population for those without severe or no Malaria infection.
A11	Full clinical update on the population for those with severe Malaria infections

physics beyond the standard model. *arXiv preprint arXiv:1807.07706*, 2018.

- [3] Anna Bershteyn, Jaline Gerardin, Daniel Bridenbecker, Christopher W Lorton, Jonathan Bloedow, Robert S Baker, Guillaume Chabot-Couture, Ye Chen, Thomas Fischle, Kurt Frey, et al. Implementation and applications of emod, an individual-based multi-disease modeling platform. *Pathogens and disease*, 76(5):fty059, 2018.
- [4] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.
- [5] Ewan Cameron, Katherine E Battle, Samir Bhatt, Daniel J Weiss, Donal Bisanzio, Bonnie Mappin, Ursula Dalrymple, Simon I Hay, David L Smith, Jamie T Griffin, et al. Defining the relationship between infection prevalence and clinical incidence of plasmodium falciparum malaria. *Nature communications*, 6:8170, 2015.
- [6] Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99(C5):10143–10162, 1994.
- [7] Casey Ferris, Benoit Raybaud, and Gregory Madey. OpenMalaria and EMOD: A case study on model alignment. In *Proceedings of the Conference on Summer Computer Simulation*, SummerSim ’15, pages 1–9. Society for Computer Simulation International. event-place: Chicago, Illinois.
- [8] Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the Future of Software Engineering*, pages 167–181. ACM, 2014.
- [9] P. Jäckel. *Monte Carlo Methods in Finance*. The Wiley Finance Series. Wiley, 2002.
- [10] Dexter Kozen. Semantics of probabilistic programs. In *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pages 101–114. IEEE, 1979.
- [11] David P. Landau and Kurt Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, 4 edition, 2014.
- [12] Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 1338–1348, Fort Lauderdale, FL, USA, 2017. PMLR.
- [13] Chikondi A. Mwendera, Christiaan de Jager, Herbert Longwe, Save Kumwenda, Charles Hongoro, Kamija Phiri, and Clifford M. Mutero. Challenges to the implementation of malaria policies in malawi. 19(1):194.

- [14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.
- [15] Tom Rainforth. *Automating Inference, Learning, and Design using Probabilistic Programming*. PhD thesis.
- [16] Boris Schäling. *The boost C++ libraries*. Boris Schäling, 2011.
- [17] T Smith, N Maire, A Ross, M Penny, N Chitnis, A Schapira, A Studer, B Genton, C Lengeler, Fabrizio Tediosi, et al. Towards a comprehensive simulation model of malaria epidemiology and control. *Parasitology*, 135(13):1507–1516, 2008.
- [18] Thomas Smith, Amanda Ross, Nicolas Maire, Nakul Chitnis, Alain Studer, Diggory Hardy, Alan Brooks, Melissa Penny, and Marcel Tanner. Ensemble modeling of the likely public health impact of a pre-erythrocytic malaria vaccine. 9(1):e1001157.
- [19] Sam Staton, Hongseok Yang, Frank Wood, Chris Heunen, and Ohad Kammar. Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 525–534. ACM, 2016.