Proceedings of the 2021 Winter Simulation Conference
S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper, eds.

DATA GENERATION WITH PROSPECT: A PROBABILITY SPECIFICATION TOOL

Alan Ismaiel
Ivan Ruchkin
Oleg Sokolsky
Insup Lee

Computer and Information Science Department
University of Pennsylvania
3330 Walnut St.
Philadelphia, PA 19104, UNITED STATES

Jason Shu

Department of Mathematics
University of Pennsylvania
209 S 33rd St.
Philadelphia, PA 19104, UNITED STATES

ABSTRACT

Stochastic simulations of complex systems often rely on sampling dependent discrete random variables. Currently, their users are limited in expressing their intention about how these variables are distributed and related to each other over time. This limitation leads the users to program complex and error-prone sampling algorithms. This paper introduces a way to specify, declaratively and precisely, a temporal distribution over discrete variables. Our tool PROSPECT infers and samples this distribution by solving a system of polynomial equations. The evaluation on three simulation scenarios shows that the declarative specifications are easier to write, 3x more succinct than imperative sampling programs, and are processed correctly by PROSPECT.

1 INTRODUCTION

Simulation experiments explore the relationship between the simulation inputs (factors) and outputs (responses). In some complex simulations, a user is interested in the statistical properties of the responses, such as a rate of an adverse event, rather than in a systematic deterministic exploration of the factors. To obtain a representative sample of the simulated responses, the user needs to ensure that the experiment designs are sampled from a desired distribution of the inputs, such as configuration parameters, initial states, and event triggers. This distribution, often discrete and temporal, needs to be sampled based on the user's intuition and miscellaneous information such as factor dependencies and probabilities of various events.

Unfortunately, the existing technologies do not allow users to flexibly and explicitly describe their intent as a distribution over the simulation inputs. Copula-based time-series models (Biller 2009) require knowing marginal probabilities and selecting a copula function, but cannot process richer probability constraints. Stochastic optimization (Valente et al. 2008) can find an optimal distribution under uncertainty, but requires an objective function. Probabilistic programming languages (PPLs) (Gordon et al. 2014) can be used to encode a sampling algorithm sequentially and imperatively. These methods are difficult to use when the user's intentions concern various probability forms (joint, marginal, conditional) and dependencies (unconditional, conditional, temporal). As a result, to explore a distribution of interest, the user is forced to implement a complex sampling algorithm, omit important information, or accept the limitations of a particular time-series model.

Consider a *motivating scenario:* an engineering team is building a controller for an autonomous cleaning vehicle that sweeps parking lanes in a city. Before deploying the controller, the team plans to test it in a high-fidelity driving simulator in the situations representative of the realistic conditions, based on the following information. A well-tested lane detector has known error rates at different times of the day. A cleaning schedule indicates the times of the day when the vehicle will be used. Finally, the history of parking tickets for street cleaning determines how often the vehicle will have to exit the lane to avoid a parked car. So, how should the team sample representative situations to test the controller? (We answer this question in Section 6.)

To address such data-generation problems, this paper introduces a modeling approach for discrete distribution in the form of *temporal declarative specifications*. These specifications have multiple ways to constrain the distribution, support arbitrary dependency structures, and do not require programming the sampling. A specification implies a probability distribution, which we determine automatically and precisely by solving a system of polynomial equations with algorithms from computational algebra. Our approach determines when the specification is inconsistent or insufficient for a unique distribution. Once the intended distribution is determined, its sampled data is returned to the user.

Our approach is implemented in an open-source tool PROSPECT (a Probability Specification Tool), found at http://github.com/bisc/prospect along with this paper's technical supplement. PROSPECT reads a textual specification language and produces a sample sequence of requested length from the specified distribution. The tool infers using the extensive algebraic libraries of the Wolfram Mathematica platform (Wolfram 2003).

We evaluate our approach on three realistic data-generation scenarios for simulators in different domains. The results show that imperatively programming a data generator demands substantial efforts from the user, both in manual probability calculations and writing non-trivial error-prone code. In contrast, PROSPECT allows for a convenient and succinct specification and automatically generates accurate data from it.

Thus, this paper makes the following contributions: (i) a *specification language* for discrete distributions, with a formal syntax and probabilistic semantics, (ii) an *algebraic inference approach* to determine a distribution from the specifications, (iii) a *software tool* PROSPECT implementing the language and the inference, and (iv) an *evaluation* of PROSPECT on three data-generation scenarios from different simulation domains.

The paper is organized as follows. Section 2 summarizes the preliminaries and formalizes our sampling problems. Section 3 describes our data generation workflow. Section 4 defines our specification language, and Section 5 details our inference. Section 6 evaluates the implementation of our PROSPECT tool. The paper concludes with related work in Section 7 and discussion in Section 8.

2 PRELIMINARIES AND PROBLEM

This section briefly summarizes the necessary probability theory, describes our treatment of independence and time, and, at the end, formulates three data generation problems to address.

This paper focuses on *discrete random variables* with *finite domains*, each modeled as a set of values $C = \{c_1 \dots c_{|C|}\}$. Infinite discrete and continuous domains are beyond this paper's scope.

Running example. A fair coin (a discrete random variable x) is tossed and lands on heads ($x = T$) or tails ($x = F$). Then, independently, a fair dice (a discrete random variable y) is tossed, resulting in outcome from 1 to 6.

Consider a set of random variables $V = \{v_1, \dots, v_{|V|}\}$. We define $\bar{V} = (v_1, ..., v_{|V|})$ to be a vector containing all elements of V arranged by some ordering method (ours is described in Section 2.1). Any subset $V' \subseteq V$ corresponds to a vector $\bar{V}'$ ordered by the same method. Each variable v_i has its set of values C_i. A trial is a random experiment that produces an *outcome*, or sample, represented by a vector of values. $\bar{C} = (c_1, \dots, c_{|V|})$ is an outcome for $\bar{V}$ if $\forall i \in [1 \dots |V|] : c_i \in C_i$. A *sample space* $\Omega(V)$ is the set of all possible outcomes for $\bar{V}$. In the running example, flipping a coin and rolling a dice is a trial, and $(T, 4)$ is a possible outcome.

Events are subsets of a sample space, tied to random variables as follows. For sample space $\Omega(V)$ and some subset $V' \subseteq V$, let $\bar{C}$ be an outcome for $\bar{V}'$. By $C(\bar{V}')$ we denote the set of all such outcomes $\bar{C}$, and so $C(\bar{V}) = \Omega(V)$ is a set of all *elementary events*. Furthermore, the notation $\bar{V}' = \bar{C}$ refers to the event of each variable in $\bar{V}'$ taking the corresponding value in $\bar{C}$. Note that this event can contain different outcomes depending on the variables whose sample space we consider. In our example, $C((x,y))$ is described by a regular expression $(T|F, 1|2|3|4|5|6)$.

A *probability distribution* $\mathbb{P}_V$ over sample space $\Omega(V)$ is a function from any event in $\Omega(V)$ to the interval $[0,1]$. For an event $e \subseteq \Omega(V)$, its probability is $\mathbb{P}_V(e)$. We define *conditional probability distributions* by conditioning some distribution $\mathbb{P}_V$ over $\Omega(V)$ on event e in the usual way. A conditional probability distribution over all events $\bar{V}' = \bar{C}$ (assuming $V' \subseteq V, \bar{C} \in C(\bar{V}')$) given event $e \subseteq \Omega(V)$ is denoted as $\mathbb{P}_V(\bar{V}' \mid e)$. For two disjoint subsets $V_1, V_2 \subseteq V$, $\mathbb{P}_V(\bar{V}_1 \mid \bar{V}_2)$ is the set of conditional probability distributions over all $\bar{V}_1 = \bar{C}_1, \bar{C}_1 \in C(\bar{V}_1)$ when conditioned on each event $\bar{V}_2 = \bar{C}_2, \bar{C}_2 \in C(\bar{V}_2)$.

Independence relations are crucial to structuring V, so we formalize two concepts of set independence.

Definition 1 (Variable Set Independence) A set of variables $V' = \{v_1 \dots v_k\}$, $V' \subseteq V$ is *independent*, denoted as $\perp V'$, if any subset of variables in V' take values independently from each other:

$$\forall j \in \{2,\dots,k\} : \forall a_1 < \dots < a_j \in \{1,\dots,k\} : \forall (c_1,\dots,c_j) \in C((v_{a_1},\dots,v_{a_j})) \ :$$

$$\mathbb{P}_V(v_{a_1} = c_1,\dots,v_{a_j} = c_j) = \prod_{i \in \{1,\dots,j\}} \mathbb{P}_V(v_{a_i} = c_i)$$

In our running example, coins are tossed independently from dice rolls: $\perp\{x,y\}$.

Definition 2 (Conditional Variable Set Independence) Given sets of variables $V_1, V_2 \subseteq V$, $|V_1| = k$, V_1 is *conditionally independent* given V_2, denoted as $\perp V_1 \mid V_2$, if events for V_1 are independent given on every possible event $\bar{V}_2 = \bar{C}, \bar{C} \in C(V_2)$:

$$\forall j \in \{2,\dots,k\} : \forall a_1 < \dots < a_j \in \{1,\dots,k\} : \forall (c_1,\dots,c_j) \in C((v_{a_1},\dots,v_{a_j})) : \forall \bar{C} \in C(\bar{V}_2) :$$

$$\mathbb{P}_V(v_{a_1} = c_1,\dots,v_{a_j} = c_j \mid \bar{V}_2 = \bar{C}) = \prod_{i=1}^{j} \mathbb{P}_V(v_{a_i} = c_i \mid \bar{V}_2 = \bar{C})$$

Our approach also requires several basic definitions from algebra (Dummit and Foote 2004).

Definition 3 (System of Polynomial Equations) A *system of polynomial equations* is a finite set of simultaneous equations $F = \{f_1 = 0, \dots, f_k = 0\}$, where $f_1 \dots f_k$ are polynomials in unknowns $X = \{x_1, \dots, x_n\}$.

Definition 4 (Solutions of Polynomial Systems) For a system of polynomial equations F in unknowns X, a *solution* $a(F)$ is an assignment of values to all unknowns in X s.t. all equations in F are true.

Definition 5 (Dimensionality, Feasibility of Polynomial Systems) An *under-determined system* has infinitely many solutions. A *well-determined system* has finitely many solutions. An *infeasible system* has no solutions.

2.1 Temporal Assumptions and Three Sampling Problems

Now we introduce a temporal structure over V to fit our data-generation scenarios of interest. We consider a model of discrete finite time, similar to discrete stochastic processes (Gallager 2013), in which a conceptual mechanism (e.g., a repeated coin flip) is represented with an indexed *family* of random variables (e.g., the coin flip outcome at each moment of time). We fix an arbitrary integer $N \geq 1$ representing the total number of time steps and define time by structuring variables V in two dimensions: by family and by time point. By family, V is split into $K \geq 1$ indexed families $V_{1,*}, V_{2,*}, \dots, V_{K,*}$. In our running example, $K = 2$.

Within each family, say $V_{1,*}$, all random variables take values from C_1 and are indexed by time: $v_{1,1}, v_{1,2}, \dots, v_{1,N}$. Similarly, $V_{2,*} = \{v_{2,1}, v_{2,2}, \dots, v_{2,N}\}$. Often, we index variables by the second dimension (e.g., $V_{*,3} = \{v_{1,3}, v_{2,3}, \dots, v_{K,3}\}$), referring to *temporal slices* through V with a meta-variable $t \in \{1 \dots N\}$. For some $V' \subseteq V$, the vector $\bar{V}'$ is sorted by ascending time and then lexicographically by variable family.

Definition 6 (State Set) For every time index t, we introduce a *State Set* B_t such that

1. There exists a *shape vector* $\bar{S} = (s_1, \dots, s_K)$ such that $\forall i \in \{1,\dots,K\} : s_i \in \{1,\dots,N\}$, and the following holds: $\forall j \in \{1,\dots,K\} : \forall k \ \in \{N - s_j, \dots, N\} : V_{j,k} \in B_N$
2. All B_t are based on the last one, B_N: $\forall v_{i,j} \in B_N : \ \forall k \in \{1 \dots N-1\} : j - k > 0 \Longleftrightarrow v_{i,j-k} \in B_k$.

Informally, for a time index t, a state set B_t is a finite set of random variables with time index before t that holds a given shape. All state sets maintain that same shape defined by the shape vector. In the running example, suppose $\bar{S} = \{1,2\}$ is the shape vector. Then B_N contains three variables: x_{N-1}, y_{N-1}, and y_{N-2}. We further define a *Window Set* to be $D_t = V_{*,t} \cup B_t$.

Definition 7 (Discrete-Time Markov Chain) A *Discrete-Time Markov Chain* (DTMC) is a discrete stochastic process that adheres to the *Markov Property*, where the conditional probability distribution of future states of the process depend only on the present state (Meyn and Tweedie 2009).

We consider two properties in the framework of DTMCs. The first is the aforementioned Markov Property: for a given time index t, we describe the state as the state set B_t such that $\mathbb{P}_V(\bar{V}_{*,t} \mid \overline{(V_{*,1} \cup \ldots \cup V_{*,t-1})}) = \mathbb{P}_V(\bar{V}_{*,t} \mid \bar{B}_t)$. The second property is an optional *Stationary Property*, which asserts that probability distributions do not change over time. Formal definitions of these properties can be found in the supplement.

The most direct sampling solution is to consider all V at once, explicitly and exhaustively define $\mathbb{P}_V$, and sample it directly. Though possible, this is impractical: V grows multiplicatively with N and K, and the number of elementary events in $C(\bar{V})$ grows exponentially with V. Moreover, in practice, $\mathbb{P}_V$ often has recurring patterns, and considering all individual variables at once would fail to take advantage of these patterns. Therefore, instead, we focus on the three simplifications below, which limit the variables to a small subset of V:

1. *Static Case:* time is irrelevant, and all sampling is *i.i.d.* The Markov Property and the Stationary Property hold. In addition, the shape vector has only zeros: $\bar{S} = \bar{0}$.
2. *Time-Invariant Case:* the distribution does not change with time, and the sampling is not independent. The Markov Property and the Stationary Property hold. The shape vector is non-zero: $\bar{S} \neq \vec{0}$.
3. *Time-Variant Case:* the distribution changes over time. The Markov Property holds, while the Stationary Property does not. The shape vector is non-zero: $\bar{S} \neq \vec{0}$.

3 DATA GENERATION WORKFLOW

Our data-generation approach goes through four high-level steps, the last three of which are fully automated:

1. The user specifies a distribution in our high-level declarative language (Section 4).
2. The specification is translated into polynomial equations (Section 5.1).
3. The system of polynomial equations is solved algebraically (Section 5.2).
4. If the solution defines a unique distribution, we sample it as a DTMC (Section 3.1). This step is described first because it determines what distributions we would want to specify.

3.1 Sampling with DTMCs and Specification Requirements

Here we show how we generate samples from distribution $\mathbb{P}_V$ by relying solely on the Markov Assumption for V. We achieve this by constructing and sampling a DTMC that represents $\mathbb{P}_V$ in each of the three cases.

For each case type, we consider an *Initialization Set* $I = V_{*,1} \cup \cdots \cup V_{*,t_i-1}$ defined by the moment t_i when B_{t_i} is the first full-sized state set: $|B_{t_i}| = |B_N|$, and $|B_{t_i-1}| < |B_N|$. Note that $t_i = \max(\bar{S})$. All three DTMCs start in a state where the variables in I are sampled, and then transition by sampling $V_{*,t}$ conditioned on their respective B_t. Let $|C(\bar{I})| = L$, $|C(\bar{B}_N)| = M$, $C'_i \in C(\bar{B}_t)$, and $C''_j \in C(\bar{I})$ for $i, j \in \mathbb{N}$. Then, the DTMCs are constructed as seen in Figure 1.

In all three cases, conditional distributions $\mathbb{P}_{D_t}(\bar{V}_{*,t} \mid \bar{B}_t)$, $\forall t \in \{1, \ldots, N\}$ are sufficient to fully define the DTMC. This includes defining the initialization stage: $\forall \bar{C}'' \in C(\bar{I})$, $\mathbb{P}_V(\bar{I} = \bar{C}'')$ is equal to a product of probabilities from the conditional distribution $\mathbb{P}_{D_1}(\bar{V}_{*,1} \mid \bar{B}_1) \ldots \mathbb{P}_{D_{t_i}}(\bar{V}_{*,t_i-1} \mid \bar{B}_{t_i-1})$. However, for each of the three cases, we also define fewer and smaller sufficient distributions. As a first step, knowing distributions $\mathbb{P}_{D_t}$ is sufficient to know all $\mathbb{P}_{D_t}(\bar{V}_{*,t} \mid \bar{B}_t)$, obtained by its conditioning. Lemmas below show two distributions sufficient for sampling in the simpler cases, as evident in the DTMCs and proven in the supplement.

Lemma 1 In the static case, a distribution $\mathbb{P}_{V_{*,i}}$ determines $\mathbb{P}_{D_t}$ $\forall t \in \{1, \ldots, N\}$.

Lemma 2 In the time-invariant case, a distribution $\mathbb{P}_{D_{t_j}}$, where $|D_{t_j}| = |D_N|$, determines $\mathbb{P}_{D_t}$ $\forall t \in \{1, \ldots, N\}$.

The time-variant case cannot be simplified in this way. Even with the Markov Property, no subset of distributions $\mathbb{P}_{D_t}$ has sufficient information to define the rest of them. Therefore, to sample the time-variant case, we are forced to specify multiple distributions. We choose $\mathbb{P}_{D_1}, \ldots, \mathbb{P}_{D_N}$ for this purpose.

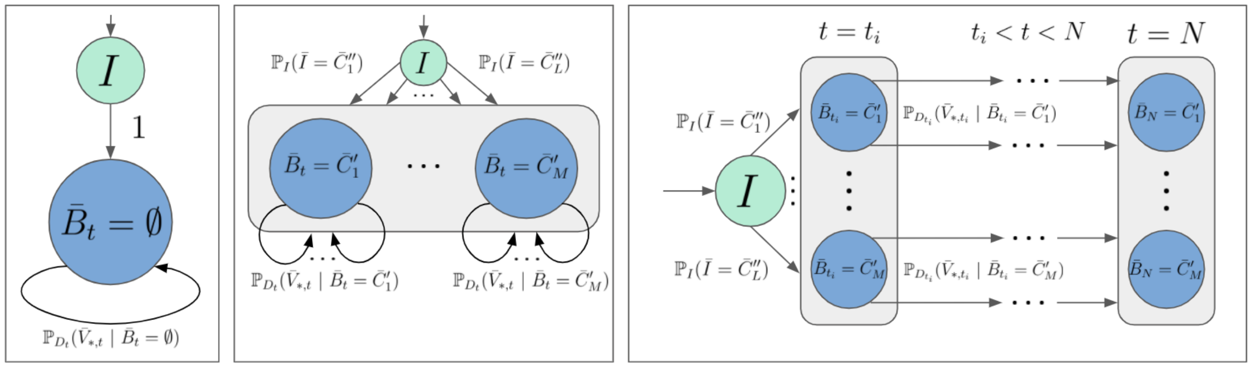

I
1
$\bar{B}_t = \emptyset$
$\mathbb{P}_{D_t}(\bar{V}_{*,t} \mid \bar{B}_t = \emptyset)$
$\mathbb{P}_I(\bar{I} = \bar{C}''_1)$
$\mathbb{P}_I(\bar{I} = \bar{C}''_L)$
$\bar{B}_t = \bar{C}'_1$
$\bar{B}_t = \bar{C}'_M$
$\mathbb{P}_{D_t}(\bar{V}_{*,t} \mid \bar{B}_t = \bar{C}'_1)$
$\mathbb{P}_{D_t}(\bar{V}_{*,t} \mid \bar{B}_t = \bar{C}'_M)$
$t = t_i$
$t_i < t < N$
$t = N$
$\bar{B}_{t_i} = \bar{C}'_1$
$\bar{B}_{t_i} = \bar{C}'_M$
$\mathbb{P}_{D_{t_i}}(\bar{V}_{*,t_i} \mid \bar{B}_{t_i} = \bar{C}'_1)$
$\mathbb{P}_{D_{t_i}}(\bar{V}_{*,t_i} \mid \bar{B}_{t_i} = \bar{C}'_M)$
$\bar{B}_N = \bar{C}'_1$
$\bar{B}_N = \bar{C}'_M$

Figure 1: From left to right, a sampling DTMC for the static, time-invariant, time-variant cases.

The next section develops a way to conveniently describe distributions $\mathbb{P}_{V_{*,t}}$, $\mathbb{P}_{D_{t_j}}$, and $\{\mathbb{P}_{D_1}, \dots, \mathbb{P}_{D_N}\}$.

4 SPECIFYING DISTRIBUTIONS

Fundamentally, the user must specify three elements of the sampling problem: the case type, the variable set V, and the distribution(s) of some window set(s) — and most of the specification devoted to the latter. Below we walk through the syntax used in the PROSPECT tool and map it to our model in Section 2.

The full specification is defined as a BNF Spec with declarations, independence, and probabilities. The declarations section determines the case, the structure of V, the desired sample count, and the random seed:

Spec ::= Decl ; Indep? ; Prob
Decl ::= 'casetype:' Case ;
'variables:' Vars ;
'values:' Vals ;
('timesteps:' NumList ;)?
'numsamples:' ⟨*int*⟩ ; ('seed:' ⟨*int*⟩ ;)?

Case ::= 'static' | 'timeinvariant' | 'timevariant'
Vars ::= StringList
Vals ::= '{' StringList (, StringList)* '}'
NumList ::= '{'⟨*int*⟩(,⟨*int*⟩)*'}'
StringList ::= '{'⟨*string*⟩(,⟨*string*⟩)*'}'

The Decl section instantiates the model parameters: Vars lists families of variables, the length of Vars defines K, the list in Vals defines C, 'timesteps' define $\bar{S}$ (optional in static case), and 'numsamples' defines N.

Variables can be referenced in three ways. In the static case, it is sufficient to use the variable's family. In the other two cases, the main probability specifications Prob are written for an arbitrary full-sized B_t at time $t > \max(\bar{S})$. Then, $x[t]$ refers to the variable of family x at time t. In the time-variant case, we also allow absolute time references, such as $x[1]$ for the variable of family x at time $t = 1$. Hence the reference syntax:

VarRefList ::= '{'VarRef (, VarRef)*'}'   VarRef ::= ⟨*string*⟩ | ⟨*string*⟩'[t'('+' | '-')⟨*int*⟩']' | ⟨*string*⟩'['⟨*int*⟩']'

The Indep section contains independence constraints for $\mathbb{P}_V$. Expressions IndepStmt and CondIndepStmt are mapped to set-based constraints in Definitions 1 and 2 respectively, which subsume the pairwise notions of independence. These constraints are applied to all times t, fixing the independence structure for $\mathbb{P}_V$.

Indep ::= 'independence''{' (IndepStmt | CondIndepStmt)
(, IndepStmt | , CondIndepStmt)* '}'

IndepStmt ::= 'indep[' VarRefList ']'
CondIndepStmt ::= 'condindep[' VarRefList, VarRefList ']'

The Prob section needs to impose sufficient constraints on $\mathbb{P}_V$ to imply a single distribution. These constraints are algebraic probability equations over probabilities of Boolean propositions over events, in which variables take/do not take the specified values. For the static and time-invariant cases, it is sufficient to define a distribution over single full-sized B_t, which we specify in the "main" part of Prob. The time-variant case, however, needs $\mathbb{P}_{D_t}$ for all $t \in \{1, \dots, N\}$, which is prohibitively difficult to do explicitly for each t.

Our solution here is *recursive specifications*: in "base", we explicitly specify the first full B_t, and in "main" we specify a recursive rule linking $\mathbb{P}_{D_t}$ to $\mathbb{P}_{D_{t-1}}, \mathbb{P}_{D_{t-2}}$, etc. Thus we intentionally restrict the scope: we will not be able to specify every possible time-variant distribution, but our specifications remain compact.

Prob ::=	StaticOrTimeInvProb | TimeVarProb	ProbExp ::=	'(' ProbExp ('+' | '-' | '*' | '/')
StaticOrTimeInvProb ::=	'main' ProbLine+		ProbExp ')' | ProbTerm | $\langle float \rangle$ | $\langle int \rangle$
TimeVarProb ::=	'base' ProbLine+ ; 'main' ProbLine+	EventExp ::=	'(' EventExp ('&&' | '||')
ProbLine ::=	ProbExp '=' ProbExp		EventExp ')' | '!(' EventExp ')' | VarVal
ProbTerm ::=	'P[' EventExp (| EventExp)? ']'	VarVal ::=	VarRef ('=' | '$\neq$') $\langle string \rangle$

Every EventExp maps to some event $e \in \Omega(V)$. Every unconditional ProbTerm maps to some $\mathbb{P}_V(e)$, and conditional ProbTerm — to some $\mathbb{P}_V(e \mid e')$ for $e' \in \Omega(V)$. Thus, every ProbLine constrains some part of $\mathbb{P}_V$.

Our running example is illustrated in Figure 2. Here, $N = 10$, $K = 2$, $|V_{*,t}| = 2$, $C = \{C_x, C_y\}$, where $C_x = \{T, F\}$, $C_y = \{1,2,3,4,5,6\}$. The $\mathbb{P}_V$ constraints include $\perp\{x,y\}$ and 6 algebraic probability constraints. Notice that $\mathsf{P}(y = 6)$ is not specified because those constraints are sufficient to determine $\mathbb{P}_V(\{x,y\})$.

```
casetype: "static"                          main
variables: {x,y}                            P[x=T]=P[x=F]
values: {{T,F}, {1, 2, 3, 4, 5, 6}}         P[y=1]=1/6
numsamples: 10                              P[y=2]=1/6
                                            P[y=3]=1/6
independence                                P[y=4]=1/6
indep[{x, y}]                               P[y=5]=1/6
```

Figure 2: Our specification for the running example.

5 INFERRING DISTRIBUTIONS

As discussed in Section 3, our approach splits the distribution into window sets D_t. Section 4 introduced a way to specify D_t, and in this section we infer a distribution $\mathbb{P}_{D_t}$ from this specification. This inference is made tractable by the simplifications of the three cases: it will be used once for the static and time-invariant cases — and twice for the time-variant case (base and recursion).

Our inference proceeds in three steps: determining the set of distribution parameters to use, translating the specification into constraints on parameters, and then solving the constraint problem.

5.1 Parameterizing Specifications

To encode probability statements as algebraic systems, we introduce *O-parameters* over the outcomes in $C(\bar{V})$.

Definition 8 (O-Parameters) Given a variable vector $\bar{V}$, each outcome $\bar{C}$ and the corresponding elementary event $e \subseteq \Omega(V)$ is assigned an *o-parameter* equal to $\mathbb{P}_V(e)$ and uniquely indexed by $\bar{C}$:

$$\forall \bar{C} \in C(\bar{V}) : \; o_{\bar{C}} = \mathbb{P}_V(\bar{V} = \bar{C})$$

Additionally, let $O(V)$ denote the set of all O-parameters for $\Omega(V)$. In the running example, $O((x,y))$ contains 12 O-parameters. E.g., for outcome $\bar{C} = (T,3)$, the parameter is $o_{(T,3)} = \mathbb{P}_V(x = T, y = 3)$.

Our strategy is to assign a set of O-parameters to every window set D_t. Then all specified constraints are translated into equations over the O-parameters. The resulting system is solved to infer the values for all O-parameters and, thus, identifying the distribution. We have three types of constraints to translate: probability lines in ProbLine, which contain ProbTerm, independence statements in Indep, and Stationary Properties.

The translations are based on lemmas (proven in the supplement), starting with ProbTerm and ProbLine:

Lemma 3 Given variables V and event $e \subseteq \Omega(V)$, $\mathbb{P}_V(e)$ can be expressed as a sum over $O(V)$.

Lemma 4 Given variables V and events $e_1, e_2 \subseteq \Omega(V)$, $\mathbb{P}_V(e_1 \mid e_2)$ can be expressed algebraically over $O(V)$.

Next, we translate independence constraints to handle IndepStmt and CondIndepStmt:

Lemma 5 Given variables V and its subset $V' \subseteq V$, an independence constraint $\perp V'$ can be equivalently translated into a finite set of algebraic constraints over parameters $O(V)$.

Lemma 6 Given variables V and its subsets $V_1, V_2 \subseteq V$, an independence constraint $\perp V_1 \mid V_2$ can be equivalently translated into a finite set of algebraic constraints over parameters $O(V)$.

Finally, we translate the Stationary Property, which is needed only in the time-invariant case:

Lemma 7 Given a set of variables V in the time-invariant case, a Stationary Property over V can be equivalently translated into a finite set of algebraic constraints over $O(V)$.

The above translations are sufficient for the static and time-invariant cases. For the time-variant case, we need a parametric way to encode a recursive dependency of $\mathbb{P}_{D_t}$ on $\mathbb{P}_{D_{t-1}}$. We observe that distribution $\mathbb{P}_{D_t}$ can be recursively and conditionally defined based on $\mathbb{P}_{B_t}$, which in turn follows from $\mathbb{P}_{D_{t-1}}$. Therefore, we introduce another set of parameters, the *Q-parameters*, to encode this recursive relationship.

Definition 9 (Q-Parameters) Given a state set B_t, each outcome $\bar{C}$ and corresponding elementary event e is assigned a *q-parameter* equal to $\mathbb{P}_{B_t}(e)$ and uniquely indexed by $\bar{C}$:

$$\forall \bar{C} \in C(\bar{B}_t): \ q_{\bar{C}} = \mathbb{P}_{B_t}(\bar{B}_t = \bar{C})$$

Being a special case of O-parameters, the Q-parameters adhere to all the previous lemmas regarding O-parameters. Similarly, $Q(B_t)$ denotes the set of all Q-parameters for B_t.

Lemma 8 Given a window set D_t and its subset $B_t \neq \emptyset$ in the time-variant case, each $q \in Q(B_t)$ is equivalent to a unique polynomial over O-parameters in $O(D_t)$.

While specs for the static and time-invariant cases are each translated into a set of numeric constraints over O-parameters, the time-variant case translates into two constraint sets: one numeric for the Q-parameters (the base case of $\mathbb{P}_{B_t}$) and one symbolic for the O-parameters in terms of arbitrary Q-parameters.

We now demonstrate the conversion to O-parameters on the static specification example from the end of Section 4, defining the distribution $\mathbb{P}_{x_t,y_t}$. The 12 O-parameters here take the form $o_{(c_x,c_y)}$ where $c_x \in C_x$ and $c_y \in C_y$. The specification is translated into the following system of 19 equations:

$$\begin{cases} \sum_{i \in C_x, j \in C_y} o_{(i,j)} = 1 \\ \forall c_x \in C_x : \forall c_y \in C_y : o_{(c_x,c_y)} = (\sum_{i \in C_x} o_{(i,c_y)})(\sum_{j \in C_y} o_{(c_x,j)}) \end{cases} \qquad \begin{cases} \sum_{j \in C_y} o_{(T,j)} = \sum_{j \in C_y} o_{(F,j)} \\ \forall c_y \in \{1,\ldots,5\} : \sum_{i \in C_x} o_{(i,c_y)} = 1/6 \end{cases}$$

5.2 Solving Algebraic Equations

The previous subsection produces sets of real-valued constraints over O-parameters. These sets are equivalent to systems of real-valued polynomial equations (defined in Definition 3) because all denominators are positive.

To algebraically solve arbitrary polynomial equations, we rely on two algorithms implemented in Mathematica: *Buchberger's algorithm* and *Cylindrical Algebraic Decomposition (CAD)*. Buchberger's algorithm generates a Gröbner basis, which can be seen as a higher-dimensional counterpart of the Gauss-Jordan elimination for linear systems (Dummit and Foote 2004). CAD finds a sequence of projections that lower the system's dimensionality to a single-variable polynomial, solves it, and lifts the solution back to the original dimensions (Jirstrand 1995). Both algorithms are guaranteed to terminate; Gröbner bases are used to determine feasibility of the system, and CAD is guaranteed to find all real solutions, though current implementations of CAD are subject to high computational complexity (Basu et al. 2006). Mathematica automatically picks an appropriate algorithm for each problem and, in general, returns solutions in complex numbers. Therefore, not every solution of an equation system is a valid set of distribution parameters: each parameter must be a real number in interval $[0,1]$.

Consider a polynomial equation system F_o over O-parameters $O(V)$. Denote the set of complex solutions of F_o as $A(F_o) \subset \mathbb{C}^{|O(V)|}$. We restrict $A(F_o)$ to its largest subset A_o such that $A_o \subset [0,1]^{|O(V)|}$. If $|A_o| > 1$, we say that our distribution is *under-specified*. If $|A'_o| < 1$, we say that our distribution is *over-specified*. If $|A_o| = 1$ with a single element $a \in A_o$, we use a to assign a value to each O-parameter and proceed with data generation. Our example has a well-determined system with this solution: $\{\forall c_x \in C_x : \forall c_y \in C_y : o_{(c_x,c_y)} \to \frac{1}{12}\}$.

6 EVALUATION

The goal of our evaluation is to compare the required manual effort, the amount of code/specification, and the accuracy of data generation between our approach and the probabilistic programming baseline in scenarios calling for non-trivial discrete distributions. We do so over three realistic data-generation scenarios with a unique distribution, one per each case. Note that the PPL baseline stands in for a broad range of imperative solutions that may take different forms (e.g., custom software tools mentioned in Section 7) but are algorithmically equivalent to a probabilistic program. To support our approach, we implemented the workflow from Section 3 in a software tool PROSPECT, the source code of which is available at http://github.com/bisc/prospect. It is also provided as a web application at https://prospect.precise.seas.upenn.edu.

6.1 Baseline Solution

For each scenario, two data-generation programs were written in the probabilistic programming language Pyro v1.5.1 (Bingham et al. 2018), based on Python v3.8.5. Both are sequential iterative algorithms that were constructed to sample a distribution described in a PROSPECT specification.

The first, *accurate* solution correctly interprets the specification and all the assumptions, step-by-step calculating the necessary probabilities and inferring the intended distribution. The code mirrors the flow of a multi-step probability calculation by hand, using Bayes' theorem, the law of total probability, and the identities of conditioning. Note that the user has to derive the non-trivial intermediate formulas either on paper or mentally to be able to write the Pyro code. We consider this baseline to be sampling the true desired distribution.

The second, *naive* solution demonstrates the possible errors when writing the sampling code by ignoring the implicit dependencies between variables. In the static case, it samples two variables in D_t by using their marginal probabilities, which incorrectly assumes their independence. In the timed scenarios, it fails to consider the dependence of a time step $V_{*,t}$ on its previous step $V_{*,t-1}$.

6.2 Scenario 1: Lane Keeping, Static Case

The motivating scenario from Section 1 is shown in Figure 3, accounting for the lane detector quality (the first four equations), the cleaning schedule (the next two equations), and the obstacle frequency (the last equation).

```
casetype: "static"                                   main
variables: {time, detection, lane}                   P[detection = "detected" | time = "day"] = .75
values: {{"day", "twilight", "night"},               P[detection = "detected" | time = "twilight"] = .4
    {"detected", "not detected"}, {"out", "in"}}     P[detection = "detected" | time = "night"] = .2
numsamples: 10000                                    P[detection = "detected" | lane = "in"] = .6
                                                     P[time = "day"] = .6
independence                                         P[time = "twilight"] = P[time = "night"]
condindep[{time, lane}, {detection}]                 P[lane = "out"] = .2
```

Figure 3: Our specification for Scenario 1.

The naive solution samples the daytime from `P(time)`, then `P(detection | time)`, and then, independently and incorrectly, the lane from `P(lane)`. The accurate solution instead derives `P(lane | detection)`. Our spec contains 12 informative lines. The reasoning part of the accurate (naive) baseline contains 32 (28) informative lines, i.e., excluding input/output, error handling, comments, and whitespace.

6.3 Scenario 2: Network Latency, Time-Invariant Case

A programmer is building a synthetic dataset to test a network latency monitor (Sinha et al. 2015) that predicts latency based on the two latest pings. This dataset is generated by a hybrid network simulator (Kiddle et al. 2003) that has configurable latency for both individual packet delays (modeled here as ping delay) and average network delays (modeled here as network latency). The programmer intends to simulate a network with a generally low latency (probability 0.8), with a semi-steady ping (probability 0.65-0.7), and a high latency that is partially observable from high ping delays (probability 0.6). We model these delays with the spec in Figure 4, which for simplicity considers two delay levels: "low" and "high".

```
casetype: "timeinvariant"
variables: {latency, ping}
values: {{"low", "high"}, {"low", "high"}}
timesteps: {0, 1}
numsamples: 10000
independence
condindep[{latency[t], ping[t-1]}, {ping[t]}]

main
P[latency[t] = "low"] = .8
P[ping[t] = "high" | ping[t-1] = "high"] = .7
P[ping[t] = "low" | ping[t-1] = "low"] = .65
P[ping[t] = "high" | latency[t] = "high"] = .6
```

Figure 4: Our specification for Scenario 2.

The naive solution starts by sampling `P(latency[t])`. Intermediate calculations derive `P(ping[t])` from `P(ping[t] | ping[t-1])`. With Bayes' Theorem, this derivation allows us to sample `P(ping[t] | latency[t])`, though this incorrectly ignores the value of `ping[t-1]`. The accurate solution, which requires a few more intermediate computations, derives `P(latency[t] | ping[t])`. Our specification for this scenario has only 10 informative lines. The accurate (naive) baseline has 31 (22) informative lines.

6.4 Scenario 3: Tool Wearing, Time-Variant Case

The gradual wear and tear of mechanical tools (e.g., for cutting, drilling, or machining) is commonly simulated using Markov models (Zhu and Liu 2018; Mor et al. 2020) to predict a tool's lifespan and test methods for tool wear estimation. It is challenging to model the gradual and partially observable deterioration.This scenario is a proper fit to our time-variant case. In the specification in Figure 5, we separate the tool's state ("functional" or "broken") from the tool's performance on a particular task ("ok" or "fail"). The tool's operation has a fixed marginal success chance (0.8), and it is implicitly linked to the tool's breakage chance, which increases over time. Initially, the tool has a high chance of remaining functional (0.9) and it decreases slightly slower if the operation is successful (0.95 vs 0.9). Once the tool is broken, it remains so.

```
casetype: "timevariant"
variables: {tool, op}
values: {{"broken", "func"}, {"fail", "ok"}}
timesteps: {1, 0}
numsamples: 100

independence
indep[{op[t], tool[t-1]}]

basecase
P[tool[0] = "func"] = .9
main
P[op[t] = "ok"] = .8
P[tool[t] = "broken" | tool[t-1] = "broken"] = 1
P[tool[t] = "func"] = .9*P[tool[t-1] = "func"]
P[op[t] = "ok" && tool[t] = "func"] =
    .95*P[op[t] = "ok" && tool[t-1] = "func"]
```

Figure 5: Our specification for Scenario 3.

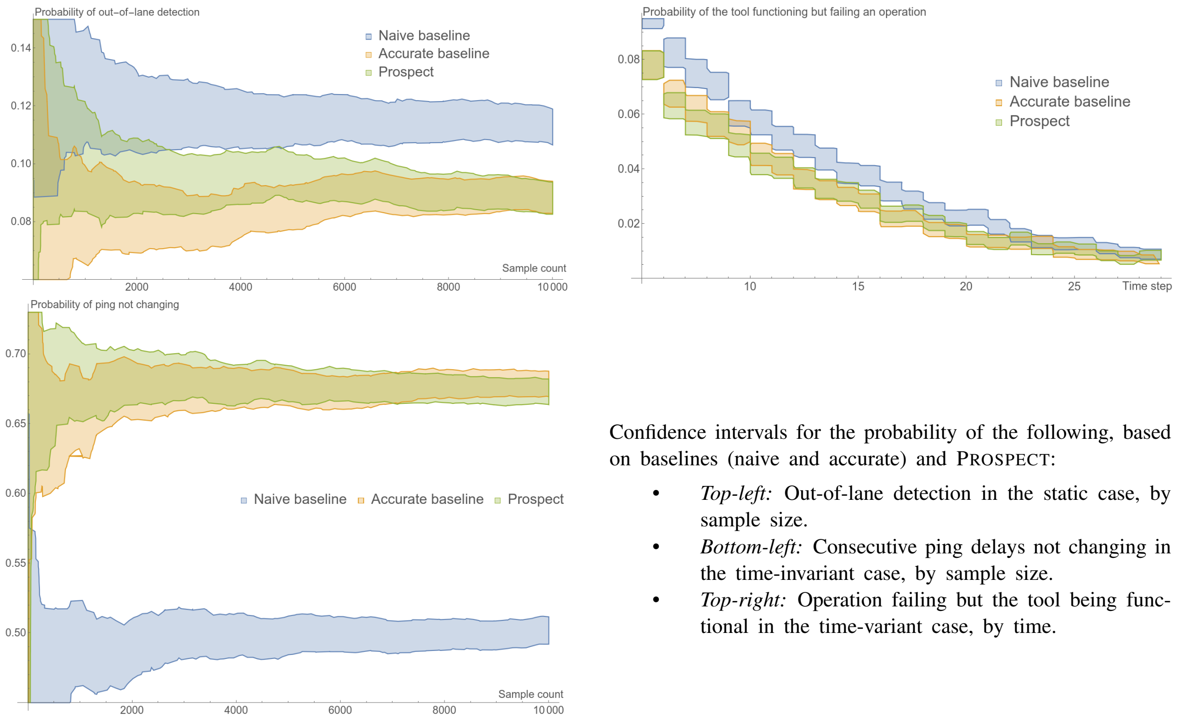
The naive solution samples `P(tool[t] | tool[t-1])` and, independently and incorrectly, samples `P(op[t])`. The accurate baseline derives a formula for `P(op[t] | tool[t])`. Our specification for this scenario has 11 informative lines. The accurate (naive) baseline has 39 (35) informative lines.

6.5 Results

The PROSPECT specs were substantially more succinct than probabilistic programs, achieving a 2–3x reduction of the informative line count. Based on our experience, the programming of the baselines took substantial cognitive effort, primarily because it necessitated finding a "path" through the probabilities by hand. Comparatively, using PROSPECT has been straightforward and only required iterations when the specification was found to be over/under-determined (which, without PROSPECT, would have been yet another cognitive task).

To evaluate accuracy, we sampled both baselines and PROSPECT, and estimated the marginal, joint, and conditional probabilities in D_t using Wilson's 95% confidence intervals (CIs) for the binomial parameter. Most probabilities turned out to be indistinguishable between the three sources, so in each scenario we highlight one of the few discrepancies — each confirmed with Pearson's chi-square test for homogeneity ($p < 0.00001$).

The probability estimates are shown in Figure 6. In all scenarios, the accurate baseline and PROSPECT were statistically indistinguishable on the full sample (10k points). In Scenario 1, the discrepancy is in `P(detection = "detected", lane = "in")`. The naive solution biased the controller testing to be overly optimistic about lane detection. In Scenario 2, a discrepancy is in `P(ping[t] = ping[t-1])`. Interestingly, the distributions within each time point were equivalent across the 3 sources, highlighting the difficulty of finding bugs in time series generators. In Scenario 3, a similar discrepancy was found in `tool[t] = "func", op[t] = "fail")`. Thus, PROSPECT has delivered accurate sampling data.



Confidence intervals for the probability of the following, based on baselines (naive and accurate) and PROSPECT:

- *Top-left:* Out-of-lane detection in the static case, by sample size.
- *Bottom-left:* Consecutive ping delays not changing in the time-invariant case, by sample size.
- *Top-right:* Operation failing but the tool being functional in the time-variant case, by time.

Figure 6: Confidence intervals for probabilities in the three sampling scenarios.

7 RELATED WORK

In the simulation context, design of experiments considers factors, their levels, and the responses obtained from the simulation (Sanchez and Wan 2015). A design is a set of tuples of factor values for which the simulation is run. Typical tasks relate factors and responses, such as finding a desired response or an area of robustness where the responses are similar. Designs are typically deterministic and based on patterns such as “grids” or “stars”. In our terms, factors, levels, and designs map precisely to variables, their values, and generated datasets. However, we consider the task to elicit an appropriate distribution of responses (e.g., car crashes), for which we require random designs that follow a desired distribution of factors. We do not explicitly consider responses; instead, we ensure that the design is drawn from the desired distribution.

Simulation designs can be captured by experiment specification DSLs (Schützel et al. 2014), such as SESSL (Ewald and Uhrmacher 2014) and NEDL (Hallagan 2011). The former is a specification language for compositional simulations, and the latter is a compact XML-based experiment description language. Such languages build deterministic designs with pre-defined patterns and provide many utilities: defining parameters, timeouts, exception handling, output format, and so on. In contrast, PROSPECT samples potentially complex random designs and thus complements the current landscape of simulation DSLs.

A variety of simulators provide high-fidelity representations of application domains, recently in cyber-physical systems: CARLA (http://carla.org) and Udacity (http://github.com/udacity/self-driving-car-sim) for cars, X-Plane (http://x-plane.com) and AirSim (http://github.com/microsoft/AirSim) for airplanes, Gazebo (http://github.com/microsoft/AirSim) for robots, and others. At a higher level of abstraction, there are discrete-event simulations, e.g., for computer networks (Hu and Sarjoughian 2005), and domain-specific DSLs, e.g., KENDRICK for epidemics (Bui Thi Mai Anh et al. 2015). Simulations at any level of abstraction can use our approach and tool to inform the selection of random experiments.

Probabilistic graphical models (Markov/Bayesian nets) are the classical ways to represent a distribution with given independence relations (Koller et al. 2009). Arbitrary independence constraints are supported in chain graphs (Lauritzen and Richardson 2002). Graphical models require full up-front specification of the distribution: conditional probabilities for Bayesian nets and potentials for Markov nets. These specifications do not necessarily match the user's intuition and do not allow the flexibility of specifying only the known aspects of the distribution. Some domains bridge this gap with intuitive domain-specific DSL, such as SSC (http://web.mit.edu/irc/ssc) for biochemical reactions that essentially specifies a DTMC. From this viewpoint, our proposed language can be understood as a domain-agnostic way to conveniently describe a graphical model.

Related to graphical models are probabilistic programming languages (PPLs) (Gordon et al. 2014), which describe distributions implicitly through their sampling algorithms. Popular PPLs like Pyro (Bingham et al. 2018) emphasize learning or inferring a model given a program and a dataset. Unlike PPLs, PROSPECT relies on an explicit declarative specification without any data, which increases the number of parameters but shortens the descriptions of complex distributions. Although PPLs can sample these exact distributions, they require extra effort and more lines of code, as Section 6 shows. Another related PPL is Scenic (Fremont et al. 2019), which samples simulation inputs but only in the context of scenarios for cyber-physical systems.

Two related areas aim to determine distributions using different inputs. First, stochastic programming optimizes objective functions over distributions and constraints. SAMPL (Valente et al. 2008) is a DSL for declarative algebraic specification of stochastic programming problems, which differ from our problem in two key ways: (i) they admit uncertain specifications, whereas we expect a single distribution; (ii) they have an objective to optimize, and we do not. Second, copula-based methods estimate a joint distribution given known marginal distributions and a copula function. The copula choice relies on expert knowledge and samples of the joint distribution. Copula-based time-series autoregressive models — ARTA, NORTA, VARTA (Biller 2009) — define random processes with known marginals. These models are distinct from PROSPECT: (i) they focus on fully-known continuous marginal distributions with few parameters and do not straightforwardly apply to exponentially-parametric discrete categorical distributions, and (ii) they require knowledge or data to choose an appropriate copula, which is not available in our scenarios.

A discrete distribution's parameters have been inferred by solving polynomial equations for Probabilistic Satisfiability (PSAT) problem (Henderson et al. 2018). The authors used Newton's method to find an approximate solution to an equation system that is equivalent to PSAT constraints. In comparison, we expand such parameterization to temporal distributions with conditional independence, create a user-facing DSL for specifying them, and use algebraic algorithms that guarantee an eventual correct answer, unlike Newton's method.

8 DISCUSSION AND CONCLUSION

The current limitations of PROSPECT's syntax and semantics point to promising future work directions. First, syntax extensions can enable limited continuous distributions and mixed expressions combining probabilities and numeric values of random variables. Second, time-invariant specifications can be extended with new patterns to cover more time-variant distributions. Finally, an exciting possibility is to extend the semantics with uncertainty in the form of compound distributions (i.e., a distribution over another distribution's parameters) and allow under-specification of distributions, which can be resolved with meta-models or tuned to data.

In conclusion, this paper introduced a declarative modeling approach for discrete probability distributions based on parameterization and solving systems of polynomials. Supported by an open-source tool PROSPECT, this approach has demonstrated its succinct, relatively low-effort, and accurate data generation. We believe that this approach could be useful not just for simulation, but also for probabilistic reasoning, design and analysis of systems, and other tasks requiring probability specification and inference.

ACKNOWLEDGMENTS

We thank Kaustubh Sridhar and Richard Avila for their feedback on the ideas in this paper. This research has been sponsored in part by the Penn Undergraduate Research Mentionship Program and the Defense Advanced Research Projects Agency as part of the Assured Autonomy program under Contract No. FA8750-18-C-0090.

REFERENCES

Basu, S. et al. 2006. *Algorithms in Real Algebraic Geometry*. 2 ed. Berlin Heidelberg: Springer-Verlag.

Biller, B. 2009, June. "Copula-Based Multivariate Input Models for Stochastic Simulation". *Operations Research* 57(4):878–892.

Bingham, E. et al. 2018. "Pyro: Deep Universal Probabilistic Programming". *Journal of Machine Learning Research*.

Bui Thi Mai Anh et al. 2015, January. "KENDRICK: A Domain Specific Language and platform for mathematical epidemiological modelling". In *2015 IEEE International Conference on Research, Innovation, and Vision for Future (RIVF)*, 132–137.

Dummit, D., and R. Foote. 2004. *Abstract Algebra*. 3rd ed., Chapter 9.6 Polynomials in Several Variables over a Field and Gröbner Bases, 315–330. University of Vermont.

Ewald, R., and A. M. Uhrmacher. 2014, February. "SESSL: A domain-specific language for simulation experiments". *ACM Transactions on Modeling and Computer Simulation* 24(2):11:1–11:25.

Fremont, D. J. et al. 2019, June. "Scenic: a language for scenario specification and scene generation". In *Proceedings of ACM Conference on Programming Language Design and Implementation*, PLDI 2019, 63–78. New York, NY, USA.

Gallager, R. G. 2013. *Stochastic Processes: Theory for Applications*. Cambridge, UK: Cambridge University Press.

Gordon, A. et al. 2014. "Probabilistic programming". In *Future of Software Engineering*, 167–181. New York, NY, USA.

Hallagan, A. 2011, January. *The Design of XML-Based Model and Experiment Description Languages for Network Simulation*. Honors Thesis, Bucknell University.

Henderson, T. et al. 2018. "Probabilistic Logic for Intelligent Systems". In *The Proceedings of the 15th International Conference on Intelligent Autonomous Systems*.

Hu, W., and H. Sarjoughian. 2005. "Discrete-event simulation of network systems using distributed object computing". In *Intl. Symp. on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Part of SummerSim*, 884–893.

Jirstrand, M. 1995. "Cylindrical Algebraic Decomposition - an Introduction". Technical Report LiTH-ISY-R-1807, Department of Electrical Engineering, Linköping University.

Kiddle, C. et al. 2003, June. "Hybrid packet/fluid flow network simulation". In *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation*, 143–152. ISSN: 1087-4097.

Koller, D. et al. 2009, July. *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: The MIT Press.

Lauritzen, S. L., and T. S. Richardson. 2002. "Chain graph models and their causal interpretations". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 64(3):321–348.

Meyn, S., and R. L. Tweedie. 2009. *Markov Chains and Stochastic Stability*. New York: Cambridge University Press.

Mor, B. et al. 2020, May. "A Systematic Review of Hidden Markov Models and Their Applications". *Archives of Computational Methods in Engineering*.

Sanchez, S. M., and H. Wan. 2015, December. "Work smarter, not harder: A tutorial on designing and conducting simulation experiments". In *2015 Winter Simulation Conference (WSC)*, 1795–1809. ISSN: 1558-4305.

Schützel, J. et al. 2014. "Perspectives on Languages for Specifying Simulation Experiments". In *Proceedings - WSC*.

Sinha, D. et al. 2015, December. "Real-time monitoring of network latency in Software Defined Networks". In *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, 1–3. ISSN: 2153-1684.

Valente, C. et al. 2008. "Extending Algebraic Modelling Languages for Stochastic Programming". *INFORMS Journal on Computing* 21(1):107–122.

Wolfram, S. 2003, August. *The Mathematica Book, Fifth Edition*. 5th edition ed. Champaign, Ill: Wolfram Media Inc.

Zhu, K., and T. Liu. 2018. "Online Tool Wear Monitoring Via Hidden Semi-Markov Model With Dependent Durations". *IEEE Transactions on Industrial Informatics* 14(1):69–78.

AUTHOR BIOGRAPHIES

ALAN ISMAIEL is an undergraduate junior in the Department of Computer and Information Science at the University of Pennsylvania. His email address is aismaiel@seas.upenn.edu.

IVAN RUCHKIN is a postdoctoral researcher in the Department of Computer and Information Science at the University of Pennsylvania. He received his Ph.D. from Carnegie Mellon University. His e-mail address is iruchkin@cis.upenn.edu.

JASON SHU is an undergraduate junior in the Department of Mathematics at the University of Pennsylvania. His email address is jasonshu@sas.upenn.edu.

OLEG SOKOLSKY is a research professor in the Department of Computer and Information Science at the University of Pennsylvania. He earned his Ph.D. from SUNY Stony Brook. His email address is sokolsky@cis.upenn.edu.

INSUP LEE is the Cecilia Fitler Moore professor in the Department of Computer and Information Science, at the University of Pennsylvania. He received his Ph.D. from the University of Wisconsin, Madison. His email address is lee@cis.upenn.edu.