A Short Survey of Stochastic Computing Models

Prometheus Peter L. Lazo[1], Francis George C. Cabarle[1]*
Jan Michael C. Yap[1,2]
[1]Algorithms & Complexity, Dept. of Computer Science,
University of the Philippines Diliman, Diliman 1101 Quezon City, Philippines.
[2]Philippine Genome Center,
University of the Philippines Diliman, Diliman 1101 Quezon City, Philippines.

ABSTRACT

In this work, several stochastic computing models are investigated on how its stochastic process was introduced. While not all existing stochastic computing models are investigated in this work, most models included in this study involve a stochastic selection process with different approaches on the application of random variables and probabilities. Some of the results of this short survey were used for the recently introduced model known as spiking Neural P systems with stochastic application of rules (⋆SN P) during the 2020 International (electronic) Conference on Membrane Computing.

1 INTRODUCTION

This study investigates the introduction of a stochastic process to computing models. A stochastic process is a set of random states indexed within a set of time parameters [24]. Theoretically, certain types of problems seem to be more solvable by algorithms that are stochastic than deterministic, albeit with a small probability of error [23]. Despite the trade-off, incorporating a stochastic process to computing models offer different advantages depending on usage, from the various applications of Markov chains, to enabling learning behavior to finite automata [4][13][1], to introducing reward and punishment to weighted grammar [21], and to modeling natural chemical reaction representations in membrane computing [15].

It is important to preface this study by stating that not all stochastic computing models in existence is not investigated but rather a particular subset of it. The interest of this work is aligned with the introduction a stochastic process to the Spiking Neural P (SN P) Systems. SN P systems is a class of P systems that incorporates the idea of spiking neurons into the area of membrane computing [6]. This model does not exhibit any stochastic behavior and as of this writing, only three SN P variants have introduced stochastic processes to the model, namely: the Stochastic Spiking Neural P (SSN P) system of [2], the Extended Spiking Neural P (ESNP) system of [27], and the Spiking Neural P system with Stochastic Application of Rules (⋆SN P) of [8].

Furthermore, another loose criterion of interest for the survey includes the application of stochastic process to selection processes. Although some models included does not meet this, hence the criterion being being loose, being those models provide meaningful insight in stochastic processes for it to be overlooked. A stochastic selection process involves two cases. The first case involves having only one option, which may seem counter-intuitive but is indeed applicable as further discussed in [8]. The other case is the sensible scenario of having multiple options. Most of the stochastic computing models included in this short survey deal with some form of stochastic selection.

This work discusses the approaches of stochastic computing models when introducing a stochastic process. The insights gathered in this short survey was referenced in the ⋆SN P model design. Section 2 discusses the definitions used throughout the study. Section 3 details the surveyed stochastic computing models along with their approaches (i.e. how random variables are introduced and usage of discrete or continuous random variables) on introducing a stochastic process. Section 4 discusses the recommendations, future work, and conclusion.

2 PRELIMINARIES

This section requires basic familiarity with *formal language*, *automata*, *computability theory*, and *membrane computing*.

2.1 Stochastic Processes and Probability

This work refers to [24] for definitions related to probability, random variables, and stochastic process. Some definition of terms in this subsection are never used in this study but are important prerequisites to clearly define probability, random variable, random vector, and stochastic process.

2.1.1 Probability. An *experiment* is any activity or procedure that may give rise to a well defined set of outcomes. A *sample space*, denoted by Ω, is the set of all possible outcomes of an experiment with a particular but unspecified outcome denoted by ω. An *event* is a subset of Ω, in which Ω is called a *certain event* while Ω^c is called an *impossible event.* Two events A and B are *disjoint* if $A \cap B = \emptyset$. An *event space*, denoted by F, is a collection of events where:

(1) $\emptyset \in F$;
(2) if $A \in F$, then $A^c \in F$;
(3) if $A_n \in F\ \forall n \in [1, \infty]$, then $\bigcup_{n=1}^{\infty} A_n \in F$.

A *probability distribution*, is a function $Pr : A \in F \to 0 \leq Pr(A) \leq 1$ that assigns the probability of an event to occur between 0 (will not occur) and 1 (will occur) where:

(1) $Pr(\Omega) = 1$;
(2) $0 \leq Pr(A) \leq 1$ for any event A;

*corresponding author: fccabarle@up.edu.ph.

(3) axiom of *countable additivity.* if $A_j \cap A_k = \emptyset$ for $j \neq k$, then
$$Pr(\bigcup_{n \in I} A_n) = \sum_{n \in I} Pr(A_n),$$
where $(A_n; n \in I)$ may be a finite or countably infinite collection of events.

A *conditional probability* is the probability of an event A given that another event B occurs denoted by $Pr(A|B)$ where $Pr(A|B) = Pr(A \cap B)/Pr(B)$. A *probability space* is a tuple (Ω, F, Pr) that fully describes an experiment from the possible outcomes Ω, to the events F and the probabilities Pr assigned to the events.

2.1.2 Random Variables. Given Ω and Pr, a *random variable* is a function $X : \omega \in \Omega \to x \in \mathbb{R}$ that assigns a real number to ω. A *discrete random variable* takes values only in some countable set $D \subset \mathbb{R}$ where typically, $D \subseteq \mathbb{Z}$. A *continuous random variable* is a random variable that takes values in an otherwise uncountable set $C \subseteq \mathbb{R}$.

2.1.3 Random Vector. Given a probability space, A *random vector* is a set of random variables $\{X_1, \ldots, X_n\}$ denoted by $\boldsymbol{X}$ with its possible values $\{x_1, \ldots, x_n\}$ denoted by $\boldsymbol{x}$.

2.1.4 Stochastic Process. Given a probability space, a *stochastic process* is a set containing random variables $X(t \in T)$ or random vectors $\{X_1(t), \ldots, X_n(t)\}$ where t is the *time-parameter* that runs over an index set T. Each element in the stochastic process takes values in some set $S \subseteq \mathbb{R}$ called the *state space* and that element is the *state* of the process at t.

2.1.5 Markov Chains. Given a state space S, a *Markov chain* is a sequence of discrete random variables where it satisfies the Markov property:
$$Pr(X_n = k|X_0 = x_0, \ldots, X_{n-1} = j) = Pr(X_n = k|X_{n-1} = j)$$
for all $n \geq 1$, and all $x_0, x_1, \ldots, j, k \in S$. If, for all n, $P(X_n = k|X_{n-1} = j) = p_{jk}$, then the chain is *homogeneous.* A *probability matrix* the an array $\boldsymbol{Pr} = (p_{jk}), j, k \in S$. A *stochastic matrix* is a probability matrix where $\sum_{k \in S} p_{jk} = 1$. A *doubly stochastic matrix* is a probability matrix where both $\sum_{k \in S} p_{jk} = 1$ and $\sum_{j \in S} p_{jk} = 1$.

2.2 Stochastic Automaton

From [4], a *Stochastic Automaton* is of the form:
$$(Y, Q, U, F, G)$$
where:

(1) $Y = \{y_1, \ldots, y_r\}$ is a finite set of *inputs*;
(2) $Q = \{q_1, \ldots, q_s\}$ is a finite set of *states*;
(3) $U = \{u_1, \ldots, u_m\}$ is a finite set of *outputs*;
(4) F is the *next state function*
$$q_{k+1} = F(y_k, q_k)$$
where $q_k \in Q$ and $y_k \in Y$; and
(5) G is the *output function*
$$u_k = G(q_k)$$

y_k, q_k, and u_k are the input, state, and output respectively at some instant k. F is stochastic and G may be deterministic or stochastic.

For each input y_α applied to the automaton at instant k, F is usually represented as a state transition probability matrix $M_k(y_\alpha)$. The (i, j)-element $p_k^{ij}(y_\alpha)$ of $M_k(y_\alpha)$ is defined by:
$$p_k^{ij}(y_\alpha) = P\{q_{k+1} = q^j | q_k = q^i, y_k = y_\alpha\}$$
where $i, j = 1, \ldots, s$ is the probability of the next state q_{k+l} being q^j when the present state q_k is q^i and the present input is y_α. For all possible next $q_j, j = 1, \ldots, s$,
$$\sum_{j=1}^{s} p_k^{ij}(y_\alpha) = 1$$
thus, $M_k(y_\alpha)$ is a stochastic matrix. The probability distribution of the state q_{k+1} is determined by the probability distribution of q_k and y_k . In the stationary case $M_k(y_\alpha)$ is not a function of k; that is, $M_k(y_\alpha) = M(y_\alpha)$ for all k. It is easily seen that a deterministic finite automaton is a special case of a stochastic automaton of which the matrix $M(y_\alpha)$ consists of zeroes and ones, and each row of the matrix contains exactly one element which is equal to unity. The function G, if it is stochastic, can also be represented in a matrix form. The (i, j)-element of the matrix is the probability of the output being u_j if the state is q^i.

2.3 Spiking Neural Networks

From [9], an *Spiking Neuron Network* N is construct that consists of:

(1) A finite directed graph $\langle V, E \rangle$ with V as the set of *neurons* and E as the set of *synapses*,
(2) $V_{in} \subseteq V$ as the subset of *input neurons*,
(3) $V_{out} \subseteq V$ as the subset of *output neurons*,
(4) for each neuron $v \in V - V_{in}$ a *threshold function* $\Theta_v : \mathbb{R}^+ \to \mathbb{R} \bigcup \{\infty\}$ where $\mathbb{R}^+ := \{x \in \mathbb{R} : r \geq 0\}$, and
(5) for each synapse $\langle u, v \rangle \in E$ a *response function* $\epsilon_{u,v} : \mathbb{R}^+ \to \mathbb{R}$ and a *weight function* $w_{u,v} : \mathbb{R}^+ \to \mathbb{R}$.

The firing of the input neurons $v \in V_{in}$ is assumed to be determined from outside of N such that the sets $F_v \subseteq \mathbb{R}^+$ of firing times (*spike trains*) for neurons $v \in V_{in}$ are given as the *input* of N. It is also assumed that a set $T \subseteq \mathbb{R}^+$ of *potential firing times* has been fixed.

A neuron $v \in V - V_{in}$ defines its set F_v of *firing times* recursively. The first element of F_v is $\inf\{t \in T : P_v(t) \geq \Theta_v(O)\}$, and for any $s \in Fv$, the next larger element of F_v is $\inf\{t \in T : t > s \textit{ and } P_v(t) \geq \Theta_v(t - s)\}$, where the *potential function* $P_v : \mathbb{R}^+ \to \mathbb{R}$ is defined by
$$P_v(t) := 0 + \sum_{u:\langle u,v \rangle \in E} \sum_{s \in F_u : s < t} w_{u,v}(s) \cdot \epsilon_{u,v}(t - s)$$
The firing times (*spike trains*) F_v of the output neurons $v \in V_{out}$ that result in this way are interpreted as the *output* of N.

The set T of potential firing times considered includes the case $T = \mathbb{R}^+$ (*SNN with continuous time*) and the case

$T = \{i \cdot \mu : i \in N\}$ for some μ with $1/\mu \in N$ (*SNN with discrete time*).

It is also assumed that for each SNN N there exists a bound $T_N \in \mathbb{R}$ with $T_N > 0$ such that $\Theta_v(x) = \infty$ for all $x \in (0, T_N)$ and all $v \in V - Y_{in}$. T_N may be interpreted as the minimum of all "refractory periods" T_{ref} of neurons in N. Furthermore, all "input spike trains" F_v with $v \in V_{in}$ are assumed to satisfy $|F_v \cap [0,t]| < \infty$ for all $t \in \mathbb{R}^+$. On the basis of these assumptions one can also in the continuous case easily show that the firing times are well-defined for all $v \in V - V_{in}$ and occur in distances of at least T_N.

For simulations between SNN's and Turing machines we assume that the SNN either gets an input (or produces an output) from $O, 1*$ in the form of a spike-train (i.e. one bit per unit of time), or encoded into the phase-difference of just two spikes. Real-valued input or output for an SNN is always encoded into the phase-difference of two spikes.

2.4 P systems

From [17], a *P system* Π is a construct

$$\Pi = (V, T, C, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m))$$

where:

(1) V is an alphabet with elements called *objects*;
(2) $T \subseteq V$ is the *output* alphabet;
(3) $C \subseteq V - T$ is set of *catalysts*;
(4) μ is a membrane structure consisting of m membranes, with the membranes (and hence the regions) injectively labeled by the elements of a given set H of m labels ($H = 1, 2, \ldots, m$); m is called the degree of μ;
(5) $w_i, 1 \leq i \leq m$, are strings which represent multisets over V associated with the regions $1, 2, \ldots, m$ of μ;
(6) An *evolution rule* is a pair (u, v) which is written in the form $u \to v$, where u is a string over V and $v = v'$ or $v = v'\delta$, where v' is a string over $a_{here}, a_{out}, a_{in} | a \in V, 1 \leq j \leq m$, and δ is a special symbol not in V. The length of u is called the *radius* of the rule $u \to v$.
$R_i, 1 \leq i \leq m$ are finite sets of *evolution rules* over V and each R_i is associated with the region i of μ; ρ_i is a partial order relation over R_i, called a *priority* relation (on the rules of R_i).

To simplify the notation, the subscript *here* for letters (objects) in evolution rules will be mostly omitted.

If Π contains rules of radius greater than one, then Π is a system with *cooperation.* Otherwise, it is a *noncooperative* system. A particular class of cooperative systems is that of catalytic systems where the only rules of a radius greater than one are of the form $ca \to cv$ or $ca \to cv\delta$, where $c \in C$, $a \in V - C$, and $v \in (V - C)^*$ Moreover, no other evolution rules contain catalysts such that there are no rules of the form $c \to v$ or $a \to v_1 c v_2$, with $c \in C$, and $a \in V - C$.

The $(m+1)$-tuple $(\mu, w_1, \ldots, w_m)$ constitutes the *initial configuration* of Π. The possibility of dissolving membranes means that the system may enter a configuration which will include only some of the initial membranes. Thus, any sequence $(\mu', w'_{i_1}, \ldots, w'_{i_k})$ with a membrane structure obtained by removing from μ all membranes different from $i_1, \ldots, i_k$ with w'_{i_j} strings over V, $1 \leq j \leq k$, and $i_1, \ldots, i_k \subseteq 1, 2, \ldots, m$, is called a *configuration* of Π. Not every configuration may be reachable through an evolution of the system and if a membrane is present in two different configurations, then it will have the same label, because labels are associated with membranes and are never manipulated during an evolution of the system.

Two configurations $C_1 = (\mu', w'_{i_1}, \ldots, w'_{i_k})$ and $C_2 = (\mu'', w''_{j_1}, \ldots, w''_{j_l})$ of Π, written as $C_1 \Rightarrow C_2$, is a *transition* from C_1 to C_2 if C_1 passes to C_2 using the evolution rules from $R_{i_1}, \ldots, R_{i_k}$ in the regions $i_1, \ldots, i_k$.

When using a rule $u \to v$ in the region i_t, copies of the objects as specified by u are removed, and the result of using the rule is determined by v.

The use of evolution rules are performed in parallel, for all possible applicable rules $u \to v$, for all occurrences of multisets u in the regions associated with the rules, for all regions, following the principles of nondeterminism and maximal parallelism.

A sequence of transitions between configurations of a given P system Π is called a *computation* with respect to Π. A computation is *successful* if and only if it halts, that is, there is no rule applicable to the objects present in the last configuration. The output of a successful computation is $\Psi_T(w)$, where w describes the multiset of objects from T sent out of the system during the computation. On the other hand, a non-successful computation has no output. The set of such vectors $\Psi_T(w)$ is denoted by $Ps(\Pi)$ ("Ps" stands for "Parikh set"), and it is *generated* by Π.

2.5 SN P systems

From [6], a *spiking neural P system* of degree $m \geq 1$ is of the form:

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0)$$

where:

(1) $O = a$ is the singleton alphabet (a is called *spike*);
(2) $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form:

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

where:
(a) $n_i \geq 0$ is the *initial number of spikes* in the neuron;
(b) R_i is a finite set of *rules* of the following two forms:
(1) $E/a^r \to a; d$, where E is a regular expression over O, $r \geq 1$, and $d \geq 0$;
(2) $a^s \to \lambda$ for some $s \geq 1$ with the restriction that $a^s \notin L(E)$ for any rule $E/a^r \to a; d$ of type (1) from R_i;
(3) $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* among neurons);
(4) $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron.*

SN P Systems function synchronously, i.e. if a neuron can fire a rule in each step, it must do so. In this way, the entire system uses a global clock to index the time. The rules of form (1) are called *firing rules*. At some step q, a neuron σ fires when it applies a firing rule $E/a^r \to a; d$ if σ contains n spikes such that $a^n \in L(E)$ and $n \geq r$. Neuron σ releases its spike at step $q + d$ to every neuron with a synapse from σ. This means that if $d = 0$ the spike leaves σ immediately while $d > 0$ means the spike will leave σ after d steps. If $d > 0$, then σ is *closed* within steps q and $q + d - 1$ where it cannot receive new spikes and consequently fire. Starting at step $q + d$ the neuron becomes *open*, i.e. it can receive new spikes and emit its spike. Any spike received by σ while it is closed is lost, i.e. the spike is never received by σ .

The rules of form (2) are called *forgetting rules*. When such rules are applied, that is when the number of spikes in a neuron $n = s$, then s spikes are removed from the neuron, i.e. they are lost, and no spike is produced. Neurons are allowed to have two firing rules $E_1/a^{r_1} \to a; d_1$, $E_2/a^{r_2} \to a; d_2$ with $L(E_1) \cap L(E_2) \neq \emptyset$. Nondeterministic application of rules occur when $a^{n_i} \in L(E_1) \cap L(E_2)$, i.e. the neuron nondeterministically chooses which such rules it fires. Note that firing and forgetting rules in a neuron cannot be applied at the same time.

A *configuration* is the distribution of spikes and the open/-closed states of all neurons. The initial configuration is $C_0 = \langle n_1, n_2, \ldots, n_m \rangle$ corresponding to initial spikes of neurons $\sigma_1, \sigma_2, \ldots, \sigma_m$, with all neurons open. Given two configurations C_1 and C_2, a *transition*, denoted by $C_1 \Longrightarrow C_2$, is the step of passing from C_1 to C_2. A *computation* is then a sequence of transitions starting in the initial configuration.

Several results can be associated with a computation. A result can be the number of spikes present in the output neuron in the halting configuration or the number of spikes sent by the neuron to the environment during the halting computation. Another result can be a finite or infinite binary string where each bit is retrieved per step from the output neuron with 1 associated to the neuron spiking and 0 if otherwise. Often the result is obtained as follows: the *time interval between the first two spikes* of the output neuron. We denote by $N(\Pi)$ the set of all natural numbers computed by Π. From the initial paper in [6] it is known that SN P systems are computationally universal, i.e. they simulate register machines, and hence also characterise NRE.

3 STOCHASTIC COMPUTING MODELS

This section compares and contrasts methods on introducing stochastic behaviour to computing models including but not limited to SN P systems. Reviewing a method considers parameters such as (1) the semantic to which the a random variable is introduced to, (2) the type of random variable, (3) the structure of the random variables, (4) the way probability values are assigned, and (5) whether probability values are static or dynamic throughout the computation. A summary of the stochastic computing models is listed in Table 1.

A probabilistic Turing machine is a type of nondeterministic Turing machine in which each nondeterministic step is called a coin-flip step and has two legal next moves [23]. The stochastic process uses discrete random variables since the possible outcome is either “heads” or “tails”. Since the stochastic process involves a selection of computation branches, each computation step uses a random vector containing two random variables for “heads” and “tails”. The probabilities for each random variable reflect the experiment of flipping a fair coin. Thus all random variables have a static probability of 0.5 which is already configured initially. A probabilistic deterministic Turing machine (PTM) is also introduced by [22] where it does not use a coin-flip step. Since it is a deterministic Turing machine, there are no nondeterministic branches to select. Instead, the probabilistic deterministic Turing machine has random state transitions. Since the set of possible states are countable, the discrete random variables are used. Also specified in the work is that a Markov chain may be associated with each PTM which means the random variables are structured as a stochastic matrix and the probabilities of those random variables are conditional.

Before proceeding to reviewing the next stochastic computing models, it is important to note that introducing a stochastic process on a selection process implies a shared probability space between random variables. Suppose the coin-flip step of the probabilistic Turing machine uses random variables where each have different probability spaces, then it is possible for both “heads” and “tails” to occur. On the other hand, as shown in PTM, a stochastic process can also determine whether a deterministic transition occurs or not, which is where plain random variables are applicable.

Proceeding to early stochastic computing models, studies on stochastic finite automata deal with introducing stochastic processes in nondeterministic transition selection. This means stochastic finite automata uses discrete random variables because of the countable possible outcomes a state could transition to. A stochastic automaton uses a stochastic matrix [4]. The random variables typically have static probabilities but it is also possible for the probabilities to be dynamic. A stochastic automaton with random environment adds a new function A to the form of the stochastic automaton to assign the updating scheme or reinforcement scheme of the model [13]. This function is used to adjust the probabilities of the random variables. Similarly, the Stochastic Transition System of [1] used a scheduler to modify the probabilities of the random variables. Dynamic probabilities extends stochastic automata to learning automata [4][13][1]. Studies on stochastic context-free grammar uses a fairly similar approach to stochastic finite automata but instead of stochastic state transitions, the stochastic process introduced to the selection of production rule used using a random vector [21].

The stochastic Petri Nets (SPN) of [12] introduces a stochastic processes differently than the previous models. Instead of using random variables to select a nondeterministic transition, random variables are used to delay the execution of a transition. Since time is the subject of the stochastic process, a continuous random variable is used rather than discrete. On the other hand, since it does not involve any selection of outcomes, a shared probability space is not required among

Table 1: Summary table of surveyed stochastic computing models.

Model	Objectives	Random Variable	Type	Structure	Probability Assignment	Static Probabilities
Spiking Neural P Systems with Stochastic Application of Rules (⋆SN P)	Introduce a stochastic process to SN P rule application	Rule application	Discrete	Random vector	Initial or preprocessed configuration, or dynamic	Static or Dynamic
Extended Spiking Neural P system (ESNPS) [27]	Approximate solutions of combinatorial optimization problems	Firing or forgetting rule application	Discrete	Random vector, stochastic matrix	OSNP guider	No
Time-free SN P Hardware Implementation using Low-Pass and High-Pass Neurons [26]	Handling stochastic loss of spikes in SN P hardware implementation	Spike loss	Discrete	Random variable	Implicit	No
Stochastic SN P (SSN P) System [2]	Create a time-free SN P System	Rule application delay	Discrete or Continuous	Random variable	Initial configuration	Yes
Multi-compartmental Gillespie algorithm evolution P systems [18]	Computational modelling tool for systems biology	Rule application	Discrete	Random vector	Multi-compartmental Gillespie algorithm	No
Dynamical Probabilistic P Systems (DPP) [19]	Modelling biological systems	Rule application	Discrete	Random vector with stochastic constant	Current multiset and rule constant based assignment	No
Probabilistic Rewriting P systems (PRP) [11]	Capturing randomness in rewriting P systems	Rule application	Discrete	Random vector	Initial configuration	Yes
Single-object-level probabilistic P System [16]	Introducing probabilities in P systems	Objects	Discrete	Unspecified	Unspecified	Unspecified
Multiplicity-of-objects-level probabilistic P system [16]	Introducing probabilities in P systems	Multiplicities of objects	Discrete	Unspecified	Unspecified	Unspecified
Communication-target-level probabilistic P system [16]	Introducing probabilities in P systems	Target membranes selection	Discrete	Unspecified	Unspecified	Unspecified
Population Dynamics P System (PDP) [3]	Create a standardized population dynamics model of ecological communities	Rule application	Discrete	Random variable	Initial configuration	Yes
K-subset Transforming Systems with Membranes [14]	Modeling light reactions during photosynthesis	Rule application	Discrete	Random variable	Compartment contents based assignment per simulation step	No
Abstract Rewriting System on Multisets (ARMS) [25]	Dealing with the overwhelming genome information	Rule application	Discrete	Random variable	Initial configuration	Yes
Stochastic Spiking Neural Networks (SSNN) [20]	Creating a nondeterministic stochastic neural model of realistic neural behaviour	Synaptic transmission execution	Discrete	Random variable	Initial configuration	Yes
Noisy Spiking Neurons [10]	Reliable digital computing with noisy spiking neurons	Neuron firing time	Continuous	Random variable	Initial configuration	Yes
Probabilistic Spiking Neuron model (pSNM) [7]	Enhance the current SNN models with probabilistic parameters	Spike arrival, PSP, firing time	Continuous	Random variable	Initial configuration	Yes
Stochastic Petri Nets (SPN) [12]	Petri nets based techniques for the quantitative analysis of systems	Transition delay	Continuous	Random variable	Initial configuration	Yes
Probabilistic Grammar [21]	Changing the manner in which a grammar is allowed to generate words	Production rule selection	Discrete	Random vector	Initial configuration	Yes
Stochastic Transition Systems [1]	Model and analyse complex stochastic behaviours	State transition selection	Discrete	Random vector	Scheduler	No
Stochastic Automaton with Random Environment [13]	Finding an optimal action out of a set of allowable actions using stochastic automata	State transition selection	Discrete	Random variable	Initial configuration then environment response	No
Autonomous Stochastic Automaton [4]	Investigating stochastic or probabilistic automata	State transition selection	Discrete	Random vector	Initial probability matrix configuration	No
Probabilistic Deterministic Turing Machine (PTM) [22]	Introduce a stochastic variant of a deterministic Turing machine	State transition	Discrete	Random variable, stochastic matrix	Initial configuration	Yes
Probabilistic Turing Machine [23]	Investigate another way of selecting a computation branch past deterministic and nondeterministic Turing machines	Computation branch selection	Discrete	Random vector	Initial configuration	Yes

the random variables in SPN. Thus it is not required to use complex structures like random vectors and stochastic matrices.

Models reviewed that are closely leading up to SN P systems include spiking neural networks and certain variants of P systems. Studies on stochastic spiking neural networks have different approaches that used random variables on different operations within the semantics of the model. The stochastic spiking neural network (SSNN) of [20] introduced stochastic synaptic transition execution and is the only approach that used a discrete random variable. On the other hand, the noisy spiking neuron model of [10] used a similar approach to the stochastic Petri nets where a continuous random variable determines the neuron firing time. Lastly the probabilistic Spiking Neuron model (pSNM) of [7] introduced stochastic processes on multiple semantics: spike arrival, post-synaptic potential, and firing time; all of which used continuous random variables. All stochastic SNN models have static probabilities.

In P systems, [16] listed four possible levels on where to introduce stochastic processes: (1) single objects, (2) multiplicities of objects, (3) rules, and (4) communication targets. Other specifics on the implementation such as the structure of the random variables, the semantic of assigning probabilities and whether it is static or dynamic are unspecified. However, since all approaches involve selection, it can be inferred that discrete random variables are used. Rule-level probabilistic P systems have been applied in works including the K-subset Transforming Systems with Membranes of [14], Abstract Rewriting System on Multisets (ARMS) of [25], and the Population Dynamics P System (PDP) of [3]. Similarities among the approach of all three rule-level probabilistic P systems include using stochastic rule application. This means multiple rules can still be applicable within a membrane. ARMS and PDP work fairly the same on different use cases where a static probability value determines whether a rule is applied or not. On the other hand, the K-subset Transforming Systems with Membranes uses dynamic probabilities set per compartment every simulation step.

Other work on stochastic P systems include the Dynamical Probabilistic P Systems (DPP) of [19], the Probabilistic Rewriting P systems (PRP) of [11], and the P systems with Multi-compartmental Gillespie algorithm evolution of [18]. DPP systems, as its name implies, uses dynamic probabilities in random application of rules similar to the work of [14], with a difference in using a random vector with a stochastic constant. The Probabilistic Rewriting P systems (PRP) of [11] which uses random vectors to select which rewriting rule to apply. Interestingly, the P system variant of [18], which implements a stochastic process similar to PRP, highlights its probability assignment process. It performs a modified stochastic simulation algorithm of [5], an extended Monte

Carlo simulation that simulates molecular dynamics, to assign its probabilities at rule-level.

It is important to note at this point that a key difference between P systems and SN P systems is that SN P systems uses only one membrane unlike P systems which can use several membrane compartments. Further more since SN P systems uses only a singleton alphabet to represent spikes, approach (1) and (2) of [16] is therefore not applicable.

Stochastic SN P systems include the Stochastic SN P (SSN P) Systems by [2], the Time-free SN P Hardware Implementation using Low-Pass and High-Pass neurons by [26], the extended SN P system (ESNPS) within the optimization SN P system (OSNPS) by [27], and the SNP Systems with Stochastic Application of Rules ($\star$SNP) by [8].

With the goal of creating time-free or asynchronous SN P systems and inspired by the stochastic Petri nets approach, SSN P systems replaced the original SN P system firing delays with random variables and added random delays to its forgetting rules as well. In effect, the activity of a SSN P neurons is governed by a stochastic process in an *a posteriori* sense. Another important aspect to note is that its stochastic process is performed every computational step, which means that the delay is dynamic every step. Unlike the previously discussed delay-related stochastic processes, SSN P systems uses both discrete and continuous random variables since the domain of the random variable is not restricted. For example the random delay can be a probability distribution or {0,1}. This means that SSN P systems allow the outcomes to be countable or otherwise

From the assumptions in [2] that SN P systems are naturally time-free, a hardware implementation of SN P systems using DRAM-based CMOS circuits was investigated in [26]. This implementation uses two basic types of neurons called low-pass (LP) neurons which contains $a \to a$ and $a^{\geq 2} \to \lambda$ as rules, and high-pass (HP) neurons which contains $a \to \lambda$ and $a^{\geq 2} \to a$ as rules. Instead of explicitly introducing a stochastic process to SN P, the hardware implementation caused a stochastic loss of spikes because with the the time-free assumption, a spike may not reach a neuron at the designed time. The work focused on handling the unreliability by introducing an error model along with the implementation. Unlike other approaches discussed in this section, the stochastic loss of spikes is treated as an effect rather than an intended incorporation to the SN P model. This means the stochastic process is not defined in the model, which includes the probability space and random variables. Nevertheless, the stochastic parameters can still be observed such that the random variables are whether the spike is lost or not; which makes it a discrete random variable. Lastly, this work highlights the importance of maintaining reliability when introducing stochastic processes.

ESNPS has a rather specific but straightforward configuration. Its neurons only contain one firing rule of form $a \to a$ and one forgetting rule of form $a \to \lambda$. These rules do not follow the specifications of firing and forgetting rules in the original SN P system model and having the same regular expression a on both firing and forgetting rule entails nondeterministic selection between the two rules. To prevent this, each rule is applied depending on a random vector of two discrete random variables with a shared probability space. The probabilities are dynamically adjusted during computation using a guider component in the OSNPS. With the modified form of rules in ESNPS, its nondeterministic behaviour becomes different from the original SN P system. As of writing, no work has been found on validating if the ESNPS approach is backwards compatible to SN P systems. Neither has been a study on proving whether including forgetting rules in the nondeterministic selection of applicable SN P rules is trivial or otherwise.

The $\star$SN P system is designed in consideration of all the models previously discussed. The objective was to introduce a stochastic process *a priori* to the application of SN P rules, which corresponds to probabilities being incorporated at rule level. There are two cases of stochastic rule application: given a number of spikes applicable to at least one neuron cases involve (1) if there is only one rule applicable, or (2) there are multiple rules applicable. In the second case, the rule applied among the several applicable rules depend on the probabilities assigned to the rule at that step. On both cases if the sum of the probabilities of all applicable rules is < 1, then there is a probability for no rule to be applied on that step, regardless of a neuron having an applicable rule. This corresponds to the neuron being idle in a given computational step. Furthermore since the neuron is not in a refractory period, it can still receive spikes on that step.

The approach of $\star$SN P differs to SSN P as the stochastic rule application delay is an *a posteriori* approach in introducing probabilities to rule application. However, both models can be combined for a variant that have both stochastic rule application and stochastic rule application delays. furthermore, $\star$SN P can still exhibit a refractory period where it cannot receive spike when a rule is applied unlike SSN P where despite having a delay, the neuron can still receive spikes.

On the other hand, both $\star$SN P and ESNP have rule-level probabilities. However, both models have different probability spaces. ESNP only reserves its probability space for its rules such that the sum of the probabilities for the rules must be 1. This means there is no probability for an ESNP neuron no have no applied neuron at any step of the computation, which differs to $\star$SN P.

4 FINAL REMARKS

This work investigated various stochastic computing models on how their stochastic process was introduced.

The models included in the survey mostly involves a stochastic selection among a set of choices such as rules or transitions. Thus the survey is limited to insights of other applications for stochastic processes that can improve deterministic models. This study recommends that in future work, other stochastic models that is not dealing with a selection process be investigated as well on how probabilities can be incorporated for uses other than selection. A model in this survey like the SSN P system with its stochastic delays is an example of this, but nevertheless the number of non stochastic selection models is few. This recommendation also leads to exploring continuous random variables as the surveyed work in this study is mostly discrete, which correlates to the nature of stochastic selection.

This study also recommends exploring more schemes or functions of dynamic probability assignment. Similar to the stochastic automata with random environment, DPP system, and ESNP system, applications to stochastic computing models comes with dynamic probabilities. Exploring more incorporations of the Gillespie algorithm to computing is also recommended.

This study is limited to understanding the approach made of introducing stochastic process to computing models and as such, investigations on reliability is recommended. Since probabilistic approaches corresponds to a probability of error, future work on reliability is of immediate concern. Other stochastic parameters such as probability distribution is also worth investigating as it is connected to reliability as well such that setting a certain probability distribution can mitigate probabilities of error.

Immediate future work of this study focuses on SN P systems and more specifically the ⋆SN P system along with the questions and recommendations discussed in [8].

REFERENCES

[1] Cattani, S., Segala, R., Kwiatkowska, M., Norman, G.: Stochastic transition systems for continuous state spaces and non-determinism. In: International Conference on Foundations of Software Science and Computation Structures. pp. 125–139. Springer (2005)
[2] Cavaliere, M., Mura, I.: Experiments on the reliability of stochastic spiking neural p systems. Natural Computing 7(4), 453–470 (2008)
[3] Colomer, M.À., Margalida, A., Pérez-Jiménez, M.J.: Population dynamics p system (pdp) models: a standardized protocol for describing and applying novel bio-inspired computing tools. PloS one 8(4), e60698 (2013)
[4] Fu, K.: 11 stochastic automata as models of learning systems. In: Mathematics in Science and Engineering, vol. 66, pp. 393–431. Elsevier (1970)
[5] Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of computational physics 22(4), 403–434 (1976)
[6] Ionescu, M., Păun, G., Yokomori, T.: Spiking neural p systems. Fundamenta informaticae 71(2, 3), 279–308 (2006)
[7] Kasabov, N.: To spike or not to spike: A probabilistic spiking neuron model. Neural Networks 23(1), 16–19 (2010)
[8] Lazo, P.P., Cabarle, F.G., Adorna, H., Yap, J.M.: A return to stochasticity and probability in spiking neural p systems. In: Pre-proc. International Conference on Membrane Computing (ICMC2020). Vienna, Austria and Ulaanbaatar, Mongolia (September 2020)
[9] Maass, W.: On the computational complexity of networks of spiking neurons. In: Advances in neural information processing systems. pp. 183–190 (1995)
[10] Maass, W.: On the computational power of noisy spiking neurons. In: Advances in neural information processing systems. pp. 211–217 (1996)
[11] Madhu, M.: Probabilistic rewriting p systems. International Journal of Foundations of Computer Science 14(01), 157–166 (2003)
[12] Marsan, M.A.: Stochastic petri nets: an elementary introduction. In: European workshop on applications and theory in Petri nets. pp. 1–29. Springer (1988)
[13] Narendra, K.S., Thathachar, M.A.: Learning automata-a survey. IEEE Transactions on systems, man, and cybernetics (4), 323–334 (1974)
[14] Nishida, T.Y.: Simulations of photosynthesis by a k-subset transforming system with membrane. Fundamenta Informaticae 49(1-3), 249–259 (2002)
[15] Obtulowicz, A.: 17 approaching a question of biologically plausible applications of spiking neural p systems for an explanation of brain cognitive functions. Research Topics in Membrane Computing: After CMC 12, Before BWMC 10 p. 45 (2012)
[16] Obtułowicz, A., Păun, G.: (in search of) probabilistic p systems. BioSystems 70(2), 107–121 (2003)
[17] Păun, G., Rozenberg, G.: A guide to membrane computing. Theoretical Computer Science 287(1), 73–100 (2002)
[18] Pérez-Jiménez, M.J., Romero-Campero, F.J.: P systems, a new computational modelling tool for systems biology. In: Transactions on computational systems biology VI, pp. 176–197. Springer (2006)
[19] Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic p systems. International Journal of Foundations of Computer Science 17(01), 183–204 (2006)
[20] Rossello, J.L., Canals, V., Morro, A., Oliver, A.: Hardware implementation of stochastic spiking neural networks. International journal of neural systems 22(04), 1250014 (2012)
[21] Salomaa, A.: Probabilistic and weighted grammars. Information and Control 15(6), 529–544 (1969)
[22] Santos, E.S.: Probabilistic turing machines and computability. Proceedings of the American Mathematical Society 22(3), 704–710 (1969)
[23] Sipser, M.: Introduction to the Theory of Computation. Cengage learning (2012)
[24] Stirzaker, D.: Stochastic Processes & Models. Oxford University Press (2005)
[25] Suzuki, Y., Fujiwara, Y., Takabayashi, J., Tanaka, H.: Artificial life applications of a class of p systems: Abstract rewriting systems on multisets. In: Workshop on Membrane Computing. pp. 299–346. Springer (2000)
[26] Xu, Z., Cavaliere, M., An, P., Vrudhula, S., Cao, Y.: The stochastic loss of spikes in spiking neural p systems: Design and implementation of reliable arithmetic circuits. Fundamenta Informaticae 134(1-2), 183–200 (2014)
[27] Zhang, G., Rong, H., Neri, F., Pérez-Jiménez, M.J.: An optimization spiking neural p system for approximately solving combinatorial optimization problems. International Journal of Neural Systems 24(05), 1440006 (2014)