Approximate Analysis of Hybrid Petri Nets with Probabilistic Timed Transitions

Hamed Ghasemieh
University of Twente, CTIT
h.ghasemieh@utwente.nl

Anne Remke
University of Twente, CTIT
University of Münster
anne.remke@wwu.de

Boudewijn R. Haverkort
University of Twente, CTIT
b.r.h.m.haverkort@utwente.nl

Gianfranco Ciardo
Iowa State University
ciardo@iastate.edu

ABSTRACT

We extend the modelling formalism of Hybrid Petri nets with so-called Probabilistic Timed Transitions (PTT), whose firing times are chosen probabilistically from a discrete and finite support. In this setting, each state of the system can have several successor states, one for each element in the discrete support of the enabled PTTs; as a consequence, the state evolution is tree-shaped. We show that with this formalism it is possible to check the validity of certain properties even when a large number of PTTs is present in the model. However, since the state evolution tree grows exponentially in the size of the potential firings of PTTs, it is impossible to traverse the entire tree even with efficient graph traversal algorithms. Hence, we resort to checking whether the probability that a certain system property holds at a given time is more or less than a given threshold. For such probabilities, we iteratively compute an approximation, based on best-first search, which can be refined by taking into account additional states, until we are able to decide whether the threshold is exceeded or not. We illustrate the feasibility of the approach on a model of a water refinery plant with cascading failures.

CCS Concepts

•Software and its engineering → Petri nets; •Theory of computation → *Timed and hybrid models; Verification by model checking;* Shortest paths;

Keywords

Hybrid Petri nets, Probabilistic timed transitions, Model checking, Approximation

1. INTRODUCTION

Many real world systems are naturally hybrid, i.e., one needs both discrete and continuous variables to realistically describe their behaviour. Over the years, several modelling formalisms have been introduced to describe and evaluate the dynamics of such systems. The most general of these formalisms are hybrid automata [2]. However, when modelling many modern applications, randomness is required; this is particularly true for critical infrastructures, where one must model the occurrence of failures and repair times in a system. There have been several approaches to this problem, each of which extends one of the conventional hybrid models with either discrete or continuous probability distributions [1, 7, 11, 18]. These works mainly differ in where the randomness is integrated into the model. One option is to replace the differential equations governing the evolution of continuous variables by stochastic versions [1]. Another option is to replace (non)deterministic jumps between system states by probability distributions [7, 18].

In the present work, we extend hybrid Petri nets [6], with so-called Probabilistic Timed Transitions (PTT), for which a firing time is probabilistically chosen from a finite real-valued set. This means that one state of the system can have several successor states, which are chosen according to the probability distribution of the respective PTT. As a result, the state evolution over time resembles a tree. To investigate whether a certain system property holds with a certain probability at a given time, we generate the state evolution tree up to that time, and accumulate the probability that the given property holds among the leaves of the generated tree.

Unfortunately, since the number of successors of each state equals the total number of possible firing times of the enabled PTTs, the state evolution tree grows exponentially. Hence, in the presence of many of such probabilistic choices, it is often infeasible to investigate the full state evolution tree (even if only up to a certain time) and to compute the exact probability for a given property. Instead, we propose to compute both, an under- and over-approximation of the probability that a given property holds and refine these approximations by iteratively taking a larger part of the state evolution tree into account, until we can decide whether the probability that the property of interest holds exceeds a certain threshold. Clearly, it is important to use efficient search algorithms to decide, iteratively, which parts of the state evolution tree to explore. In this paper, we assign scores to

VALUETOOLS 2015, December 14-16, Berlin, Germany
DOI: 10.4108/eai.4-1-2016.151093

nodes of the state evolution tree, based on current knowledge, i.e., how much probability they attribute, and based on a prediction for the future, i.e., an estimation based on the property of interest. Then, we use an instance of a best-first search algorithm to expand nodes with a high score early in the computation [16].

This paper is organized as follows. In Section 2 we briefly review the related literature. Section 3 recalls and extends the modeling formalism of hybrid Petri nets with PTTs. Section 4 discusses the generation of the state space, and Section 5 investigates how to efficiently explore it to check system properties. Finally in Section 6 we provide a case study to show the feasibility of the approach.

2. RELATED WORKS

Hybrid Petri nets form a restricted subclass of hybrid automata where all occurrences of non-determinism are resolved by probabilities [5]. The drifts are restricted to be piecewise deterministic but we still allow the discrete and continuous parts to control each other. When adding random behaviours through PTTs, we are able to compute approximated results for this model class. This is often useful since the verification of (probabilistic) hybrid automata has been shown to be undecidable except for some restricted subclasses [2, 18]. Similar to abstraction techniques proposed for the restricted class of probabilistic timed automata [13], and probabilistic hybrid automata [20], we obtain upper and lower bounds when analysing hybrid Petri nets with PTTs. However, the approach presented in [20] is quite different from ours, as they compute a fixed upper and lower bound for a given property and point in time, which cannot easily be refined.

Recent works have extended Hybrid Petri nets with transitions that fire according to a continuous probability distribution, so-called general transitions [9]. But in practice, results only can be computed for model instances with up to two general transitions [7]. By replacing general transitions with PTTs we move from continuous to discrete support of probability distributions. This eases computation and allows us to deal with many more probabilistic transitions.

The idea of choosing firing times of transitions from a finite set is not new. Integer Timed Petri nets [19] are ordinary Petri nets where transition firing times are chosen from a finite integer-valued set. As this framework does not include continuous variables, and since state changes take place at integer times, it is possible to use decision diagrams for state representation and reachability analysis. However, the same does not hold for hybrid systems. Moreover, the idea of generating relevant portions of the state space with respect to the measure of interest has also been exploited for analytical solution of large Markov models [12, 15, 17].

3. HYBRID PETRI NETS WITH PTT'S

We extend the formalism of hybrid Petri nets of [8] to describe real systems containing both discrete and continuous variables, combined with *discrete probabilistic* behaviour. A hybrid Petri net consists of three main sets of components: (i) *places* (discrete and continuous), which model different modes or states of a system, (ii) *transitions* (discrete and continuous), which allow both probabilistic and deterministic changes between different modes of the system, and (iii) *arcs* (connecting places and transitions), which determine how the other two sets are related, i.e, how a transition between different modes (states) of the system can take place. Figure 1 shows the graphical representation for these three components. Below, we describe these in more details.

We define a hybrid Petri net with probabilistic timed transitions as a tuple $(\mathcal{P}, \mathcal{T}, \mathcal{A}, \Phi)$, where $\mathcal{P}$ is the set of places, $\mathcal{T}$ is the set of transitions, $\mathcal{A}$ are the arcs, and Φ is a tuple of functions. The set $\mathcal{P}$ is partitioned into $\mathcal{P}^D$ and $\mathcal{P}^C$, the discrete and continuous places, respectively. The former keeps track of discrete variables in the system, e.g., the number of spare parts, and the latter describes the continuous state of the system, e.g., the amount of fluid in a container. A discrete place may contain a number of *tokens*, while a continuous place is assigned a real number, representing the level of fluid residing in it. We later refer to the contents of all the places as the *marking*.

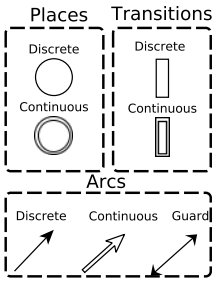

Figure 1: Graphical representation of hybrid Petri net primitives.

Transitions trigger a change in the state of the system, i.e., they may change the content of place(s), provided that all the required resources are available. In that case we say that the transition is *enabled* and may *fire*. The set $\mathcal{T}$ is also partitioned into two sets: discrete transitions, $\mathcal{T}^D$, and continuous transitions, $\mathcal{T}^C$. Continuous transitions change the contents of continuous places, and as their name suggests, continuously change the contents of continuous places connected to them according to an assigned *nominal rate*, whenever they are enabled. Discrete transitions change instead the contents of the discrete places in the system. These transitions embody the probabilistic nature of the system, and we call them *probabilistic timed transitions (PTTs)*.

A PTT a is defined by its potential firing times and their corresponding probabilities: $\mu = (t_1^a{:}\alpha_1^a, t_2^a{:}\alpha_2^a, \cdots, t_{m_a}^a{:}\alpha_{m_a}^a)$, where $t_i^a \in \mathbf{R}^+$ is a possible firing time of a, $\alpha_i^a \in [0,1]$ is the corresponding probability of firing, and m_a is the number of possible firing times for a; we call μ the *firing density* of the PTT. Moreover, we have $0 \leq t_1 < t_2 < \cdots < t_{m_a} < \infty$, and $\sum_{i=1}^{m_a} \alpha_i^a \leq 1$ which means that a may not fire at all with probability $1 - \sum_{i=1}^{m_a} \alpha_i^a$. This definition encompasses two well-known and widely-used transition types, *immediate* and *(deterministic) timed transitions*. The former corresponds to the firing density, (0:1), the latter to (t:1). Formally, each PTT a is associated with a clock c_a, which evolves with drift $dc_a/dt = 1$ whenever the transition is enabled. When the clock reaches one of the firing times t_i^a, we say that the transition a has *concession* to fire, and, if it does, its clock is reset to zero. Note that the associated clock of a PTT evolves when it is enabled, and keeps its value while it is disabled.[1]

The set of *arcs* $\mathcal{A}$ defines how transitions and places are connected to each other. Discrete arcs, collected in the set $\mathcal{A}^D$, connect discrete places to transitions as follows. If a

[1] This notion of associating a clock with each transition is different from well-known works like [14], in which transitions firings are coordinated with respect to a globally maintained clock.

transition fires, it removes tokens from places connected to it via input arcs, and adds tokens to the places that are connected via output arcs. The number of tokens removed or added is determined by a number assigned to the discrete arcs. Continuous arcs, collected in $\mathcal{A}^C$, connect continuous places to transitions. While a continuous transition is enabled, it removes contents from its input places and adds it to the contents of its output places, with a specific rate assigned to the transition. Finally, the set of guard arcs $\mathcal{A}^G$ connects discrete transitions to both discrete and continuous places. These arcs ensure that a transition is enabled only if the connecting place fulfils a threshold condition specified on the guard arc, e.g., if the amount of fluid in a connecting place is more (or less) than a constant in case of a continuous place, or if the number of tokens is more (or less) than a given value, in case of a discrete place.

The tuple $\Phi = (\phi_b^{\mathcal{P}}, \phi_p^{\mathcal{T}}, \phi_d^{\mathcal{T}}, \phi_c^{\mathcal{T}}, \phi_n^{\mathcal{A}}, \phi_u^{\mathcal{A}}, \phi_s^{\mathcal{A}}, \phi_p^{\mathcal{A}})$ contains eight *functions*. Function $\phi_b^{\mathcal{P}} : \mathcal{P}^C \to \mathbf{R}^+ \cup \{\infty\}$ assigns an upper bound to each continuous place, function $\phi_p^{\mathcal{T}} : \mathcal{T}^D \to \mathbf{N}$ specifies the priority of each discrete transition to resolve firing conflicts, as is discussed later. The mapping $\phi_d^{\mathcal{T}} : \mathcal{T}^D \to \mu$ assigns a firing density (discrete probability measure) $\mu : S \cup \{\infty\} \to [0, 1]$ to each discrete transition (PTT), as described earlier, where S is a finite countable set as the support of firing times of the PTT, and $\sum_{s \in S \cup \{\infty\}} \mu(s) = 1$. Explicitly including ∞ allows for the possibility that the transition never fires. Continuous transitions have a constant nominal flow rate defined by $\phi_c^{\mathcal{T}} : \mathcal{T}^C \to \mathbf{R}^+$. We assign an integer to each discrete input or output arc: $\phi_n^{\mathcal{A}} : \mathcal{A}^D \to \mathbf{N}$, to define the number of tokens taken from, or added to, the corresponding place upon the firing of the transition. $\phi_u^{\mathcal{A}} : \mathcal{A}^G \to \{(\triangleright, \mathbf{R})\}$, with $\triangleright = \{\geq, <\}$ assigns a comparison operator to a real number to each guard arc. The functions $\phi_s^{\mathcal{A}}, \phi_p^{\mathcal{A}}$ specify the share and the priority of a static continuous transition, as explained in the following.

Finally we address rate adaptation and race policies. Continuous transitions with concession are enabled and continuously transport fluid along fluid arcs. Conflicts in the distribution of fluid occur when a continuous place reaches one of its boundaries. To prevent overflow, the fluid input has to be reduced to match the fluid output; to prevent underflow, the fluid output has to be reduced to match the fluid input. The nominal firing rate of a continuous transition is then modified according to the *share* $\phi_s^{\mathcal{A}} : \mathcal{A}^C \to \mathbf{R}^+$ and *priority* $\phi_p^{\mathcal{A}} : \mathcal{A}^C \to \mathbf{N}$ assigned to the continuous arcs connecting the transition to the places. This is done by distributing the available fluid over all continuous arcs. Those with highest priority are considered first and if there is enough fluid available, all transitions with the highest priority can still move fluid at their nominal speed. Otherwise, their *actual fluid rates* are reduced according to the firing rate of the connected transitions and the share of the arc, as in [6]. The change of fluid rates in these cases results in a piecewise constant evolution of continuous variables present in the system, i.e., the fluid derivative of continuous places.

It is possible that several discrete transitions have concession to fire at the same time. In such situations we need to make sure that at each time point at most one transition fires. This can be done by altering the firing probabilities of these transitions, based on their priorities given by function $\phi_p^{\mathcal{T}}$. More formally, let G^t be the set of all enabled transitions with positive firing probability at time t, i.e., $a \in G^t$ if and only if $\phi_d^{\mathcal{T}}(a)(t) = \mu_a(t) > 0$. For a transition $a \in G^t$, let p_a be its firing probability at time t. Then a will fire with probability:

$$\hat{p}_a = \left(\frac{\phi_p^{\mathcal{T}}(a)}{\sum_{b \in G^t} \phi_p^{\mathcal{T}}(b)} \right) p_a. \quad (1)$$

Since transitions fire probabilistically, a transition may not fire at all, despite having concession. In this case, its clock continues, whereas, after firing, its clock is reset to zero.

4. STATE EVOLUTION

Markings, i.e., the content of places, are collected into two vectors, the discrete marking $\mathbf{m} = (m_1, \ldots, m_{|\mathcal{P}^D|})$ and the continuous marking $\mathbf{x} = (x_1, \ldots, x_{|\mathcal{P}^C|})$. The initial marking is composed of a discrete part $\mathbf{m}_0$ describing the initial number of tokens in the discrete places and a continuous part $\mathbf{x}_0$ describing the initial amount of fluid in the continuous places.

4.1 System state

The overall *state* of the model is $\Gamma = (\mathbf{m}, \mathbf{x}, \mathbf{c}, \mathbf{d})$, where the vector $\mathbf{c} = (c_1, \ldots, c_{|\mathcal{T}^D|})$ contains a clock c_i for each discrete transition representing the time that T_i has been enabled. When a transition is disabled, its clock does not evolve, but its clock value is preserved until the transition is enabled again. Clocks are only reset when the corresponding discrete transition (PTT) fires. Vector $\mathbf{d} = (d_1, \ldots, d_{|\mathcal{P}^C|+|\mathcal{T}^C|})$ indicates the drift of all continuous variables. For continuous places it indicates the change of fluid per time unit, and for discrete transitions it is the clock drift, which is either one or zero for enabled and disabled transitions, respectively. Even though $\mathbf{d}$ is uniquely determined by $\mathbf{x}$ and $\mathbf{m}$, in combination with the condition of guard arcs, it is included in the definition of a state for ease of discussion. The initial state of the system is $\Gamma_0 = (\mathbf{m}_0, \mathbf{x}_0, \mathbf{0}^{|\mathcal{T}^D|}, \mathbf{d}_0)$, where $\mathbf{0}^m$ is the vector with m zero elements.

4.2 State evolution tree

In [7, 8], the value of stochastic variable(s) characterizing the firing time of general transitions was chosen from a dense set and the state space was generated by partitioning and forming regions with similar properties, over the values of these dense sets. This setting allows us to have general transitions that can fire according to arbitrary continuous probability distributions, but the number of stochastic variables that can be handled in practice is at most two. The advantage of using PTTs is instead that the firing times are chosen from discrete sets. Hence, going from one state to another can be interpreted as a probabilistic jump. This comes with restrictions on the probability distributions, but it avoids the constraint on the number of stochastic variables present in the system. In this section, we analyse the state evolution of the system over time, and provide a method based on effective search algorithms for generating and exploring the state evolution.

In each state of the system, three types of potential events can occur: (i) a continuous place reaches its lower or upper boundary, (ii) a continuous place reaches the weight of the guard arc connected to it, and (iii) an enabled discrete transition fires. Event type (i) imposes a change in the drift of the continuous place due to rate adaptation [6]. Event type (ii) may enable or disable a transition: if the transition is

discrete, its clock drift is set to either one or zero; if the transition is continuous, the drifts of the connected continuous places can change. Hence, both event types (i) and (ii) alter the drift vector $\mathbf{d}$. Instead, event type (iii) alters the discrete marking $\mathbf{m}$.

While the first two event types are deterministic, the third type is a probabilistic choice between several events, therefore the next state is determined probabilistically, i.e., each state may have several successors, each of them chosen with a certain probability. This suggests that the *state evolution* of the system over time forms a directed tree structure where each node represents a state and each edge an event leading from the current state to the next state of the system. Each node embodies the state Γ of the system as introduced in previous section, together with the time at which the system can enter that state, while each edge is associated with the probability of occurrence of the corresponding event. For each node u, we denote its state and time of entry by $\Gamma(u)$ and $t(u)$, respectively. Figure 2 shows a schematic state evolution of the system, where u_0 is the root of the tree, i.e., the node with the initial state Γ_0, at time $t = 0$, i.e., $\Gamma(u_0) = \Gamma_0$ and $t(u_0) = 0$. For each node u_i the probability of going from that node to node u_j s denoted by $p(u_i \to u_j)$. This is the probability associated with the event that takes the system from $\Gamma(u_i)$ to $\Gamma(u_j)$.

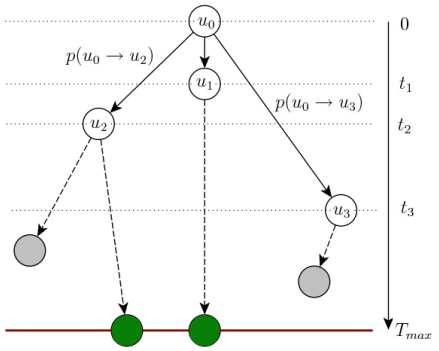

Figure 2: A state evolution tree w.r.t. a maximum time $T_{\max}$.

As stated earlier, event types (i) and (ii) are deterministic and occur with probability 1. Hence, a node for which the next occurring event is deterministic has only 1 successor. For event type (iii), instead, we have several successors, each of which occurs according to the probability of the corresponding event. To resolve deterministic events, we iterate over all continuous places and check whether they can reach their upper or lower limits, type (i), or the weight of a corresponding guard arc, type (ii), before occurrence of an event of type (iii). In the following, we focus on probabilistic events.

With each node, we associate the time when the system entered the corresponding state. The occurrence of each event advances the system in time. Figure 2 shows the evolution of time vertically, each horizontal dashed line depicts the time associated with the node(s) on it.

We now calculate the probability of moving from one node to another by firing a PTT. Assume that, at node u, the set S_e^u contains all enabled transitions, and, for each $a \in S_e^u$, let $(t_1^a{:}\alpha_1^a, \cdots, t_k^a{:}\alpha_k^a, \cdots, t_{m_a}^a{:}\alpha_{m_a}^a)$ be the associated firing density. Let $m_a(t)$ be the index of the last time point at which transition a may fire before time t, i.e., $m_a(t) = \max\{1 \le k \le m_a : t_k^a < t\}$. Then the probability that an arbitrary $b \in S_e^u$ fires at time t_i^b and takes the system to node u_i^b equals the probability that none of the enabled transitions fires before time t_i^b and that transition b fires at time t_i^b (with probability α_i^b). To compute the probability that no transition fires before time t_i^b, we need to multiply all the complementary probabilities of firing of all the enabled transitions. Hence we have:

$$p(u \to u_i^b) = \alpha_i^b \cdot \prod_{a \in S_e^u - \{b\}} \left(1 - \sum_{k=1}^{m_a(t_i^b)} \alpha_k^a\right). \qquad (2)$$

The product considers all enabled transitions $a \in S_e^u$, except b itself, and computes the probability that none of these fire before time t_i^b. If several transitions can fire at the same time, the computed probability in the above equation is distributed among them, with the strategy given in (1), where p_a is replaced by $p(u \to u_i^b)$ and G^t by $G^{t_i^b}$.

Now, let node u be reachable from u_0 along path $u_0 \rightsquigarrow u$. Then, since the path between each two nodes is a unique, the probability of being in u, $p(u)$, is given by the product of the probabilities of all the edges on the path:

$$p(u) = \prod_{(u_i, u_{i+1}) \in u_0 \rightsquigarrow u} p(u_i \to u_{i+1}). \qquad (3)$$

Algorithm 1 shows the procedure to expand a given node u, i.e., generate the set of its successors $\mathcal{N}_u$, until a maximum analysis time $T_{\max}$. Lines 1-2 initialize the final set $\mathcal{N}_u$ to the empty set and the time to the next event Δt_{next} to the distance to the maximum analysis time. Lines 3-4 iterate over all continuous places to find deterministic events of type (i) and (ii). The next event of one of these types is the one that can occur first, at time Δt_{next}. Note that Δt_P, the time to the event associated with place P, can be computed based on the drift of P and its current fluid content by solving a simple linear equation. Then, we iterate over all enabled PTTs (line 6), and for each one we consider those firing times smaller than Δt_{next}, the time of occurrence of the next deterministic event (line 7). As discussed earlier, each possible firing time of a PTT introduces a new successor to u. For each transition a and its possible firing time t_i^a, lines 8-9 create a node u_i^a and update its state and entry time based on the state and the entry time of u. This means that transition a is fired and, based on the new marking, rates of all other continuous transitions are updated, while the new content of continuous places is computed and updated up to time t_i^a, based on their previous drift in the marking of node u. Line 10 computes the probability of being in u_i^a based on Equations (2) and (3). Finally, the new node is added to $\mathcal{N}_u$, the set of successors of u.

As mentioned earlier, it is possible that none of the PTTs with concession fires before time Δt_{next}, in which case the next deterministic event occurs at the previously computed time Δt_{next}. If no deterministic event can occur before time $T_{\max}$, a special node $u_{\max}$ is created, which corresponds to reaching the maximum time horizon. This is done in lines 13-16, and probability $1 - p_{\text{total}}$ (where p_{total} is the total probability of firing of PTTs) is assigned to the edge leading to $u_{\max}$. Note that $T_{\max}$ determines the depth of the state evolution tree, hence influences its shape since, for each node, only those events that can happen before $T_{\max}$ are going to be considered. In Figure 2, for instance, if $T_{\max}$ is less

Algorithm 1 *expand*$(u, T_{\max})$

Require: Node u to expand, and $T_{\max}$ the maximum time
Ensure: The set $\mathcal{N}_u$ containing the successors of u

$\mathcal{N}_u \leftarrow \emptyset$
$\Delta t_{\text{next}} \leftarrow T_{\max} - t(u)$
for all $P \in \mathcal{P}^C$ **do**
    $\Delta t_{\text{next}} \leftarrow \min(\Delta t_{\text{next}}, \Delta t_P)$
      {Δt_P is the time to the associated event with P}
$p_{\text{total}} \leftarrow 0$
for all $a \in S_e^u$ **do**
    for all $0 \leq i \leq m_a$ s.t $t_i^a < \Delta t_{\text{next}}$ **do**
        $t(u_i^a) \leftarrow t(u) + t_i^a$
        $\Gamma(u_i^a) \leftarrow \text{update}(\Gamma(u), t_i^a)$
          {fires a and updates the state }
        $p(u_i^a) \leftarrow p(u).p(u \rightarrow u_i^a)$
        $\mathcal{N}_u \leftarrow \mathcal{N}_u \cup u_i^a$
        $p_{\text{total}} \leftarrow p_{\text{total}} + p(u_i^a)$
$t(u_{\max}) \leftarrow \Delta t_{\text{next}}$
$p(u_{\max}) \leftarrow 1 - p_{\text{total}}$
$\Gamma(u_{\max}) \leftarrow \text{update}(\Gamma(u), \Delta t_{\text{next}})$
$\mathcal{N}_u \leftarrow \mathcal{N}_u \cup u_{\max}$
return $\mathcal{N}_u$

than t_3, then node u_3 will not be generated and is replaced by $u_{\max}$. Hence, the leaves of the tree are going to be those nodes corresponding to reaching the maximum time, except for the case that an event occurs exactly at time $T_{\max}$.

Even though we build the state evolution tree for a finite time bound and the support of each PTT is finite, there still is the possibility of having an infinite number of nodes. This happens whenever an infinite sequence of *vanishing markings* occurs, i.e., a sequence of markings where a PTT representing an immediate transition is enabled. This problem is well-known for all Petri nets formalism that allow immediate transitions. However, we require that models be bounded, i.e., the number of tokens is limited, infinite sequences of vanishing markings can only take place in the form of *cycles* of vanishing markings, and these cycles can be detected and removed [3]. This ensures that we always reach a *tangible marking*, i.e., a marking in which no immediate transition is enabled, in a finite number of steps. In other words, the number of nodes in the state evolution tree with a finite time bound is always finite; therefore, for a bounded model and a finite time bound, the algorithm always terminates.

5. EXPLORING THE STATE EVOLUTION

This section describes the system properties whose validity we want to investigate. It also discusses the generation and exploration of the state evolution tree in an effective way with respect to these measures of interest.

5.1 Measures of interest

We are interested in computing the probability that a certain property is satisfied in the system at a given time τ. For this we will use the same logic as in [8]:

$$\Psi = \neg\Psi \mid \Psi \wedge \Psi \mid n_i = k \mid x_j \leq c, \tag{4}$$

where, n_i is the number of tokens in the discrete place P_i, x_j is the fluid level in the continuous place P_j, k is an integer, and c is a real number.

Let $\mathcal{U}^\tau$ be the set of all nodes the system can be in at time τ. Hence, the probability of formula Ψ being satisfied at time τ, can be computed as:

$$\pi^\Psi(\tau) = \sum_{u \in \mathcal{U}^\tau} \mathbb{1}_\Psi(u) p(u) = \sum_{u \in \mathcal{U}^\tau} \pi_u^\Psi(\tau), \tag{5}$$

where $\mathbb{1}_\Psi(u)$ is the indicator function, which equals one if Ψ holds for u and zero otherwise. Moreover, we use $\pi_u^\Psi(\tau) = \mathbb{1}_\Psi(u)p(u)$ to show the amount of probability contributed by node u to the property Ψ at time τ.

The exact computation of $\pi^\Psi(\tau)$ is only possible if we determine the set $\mathcal{U}^\tau$, which in turn is only possible if we explore the entire state evolution tree. In the next section we show how we can instead approximate its value by visiting only a portion of the state evolution tree.

5.2 Exploration

The problem of finding the possible states of the system at a given time τ can be reduced to a tree search problem where the goal nodes are those where the system can reside in at time τ. In our framework, this is equivalent to generating the state evolution until $T_{\max} = \tau$. Consequently, $\mathcal{U}^\tau$, in (5) contains leaves of the generated state evolution tree. Figure 2 depicts these goal nodes in green.

Unfortunately, the state evolution tree grows exponentially in the number of PTTs and their corresponding number of firings. More precisely, the branching factor at node u is given by:

$$b_u = \left(\sum_{a \in S_e^u} m_a\right) + 1 \leq \left(\sum_{a \in \mathcal{T}^D} m_a\right) + 1 = b_{max}, \tag{6}$$

i.e., the total number of possible firings of enabled transitions plus one, to include the possibility that no transition fires. The upper bound of this branching factor occurs when all the PTTs are enabled. Let δ be the minimum time distance between occurrence of two consecutive events. Then, the corresponding tree has at most τ/δ levels. Hence, the total number of nodes in the state evolution tree is $O(b_{max}^{\tau/\delta+1})$. Without an effective method for choosing and expanding the nodes, exploring the entire state evolution for a large number of firing times is infeasible.

Since, in practice only a portion of the state evolution can often be traversed, we compute an approximation of the probability that a given formula holds as follows. Denote the accumulated probabilities that formula Ψ and its negation $\neg\Psi$ hold at time τ by $\hat{\pi}^\Psi(\tau)$ and $\hat{\pi}^{\neg\Psi}(\tau)$, respectively. We have $\hat{\pi}^\Psi(\tau) \leq \pi^\Psi(\tau)$ and $\hat{\pi}^{\neg\Psi}(\tau) \leq \pi^{\neg\Psi}(\tau)$, where the equality holds if we explore the entire state evolution until time τ. Thus, we have $\hat{\pi}^\Psi(\tau) + \hat{\pi}^{\neg\Psi}(\tau) \leq 1$. This means that $\hat{\pi}^\Psi(\tau)$ and $1 - \hat{\pi}^{\neg\Psi}(\tau)$ provide a lower and upper approximation for $\pi^\Psi(\tau)$, respectively:

$$\hat{\pi}^\Psi(\tau) \leq \pi^\Psi(\tau) \leq 1 - \hat{\pi}^{\neg\Psi}(\tau). \tag{7}$$

If we seek to investigate whether the probability that Ψ holds at time τ is at least a given threshold q, we can stop the exploration if either $\hat{\pi}^\Psi(\tau) \geq q$ or $\hat{\pi}^{\neg\Psi}(\tau) > 1 - q$. In other words, we cannot stop as long as:

$$\hat{\pi}^\Psi(\tau) \leq q < 1 - \hat{\pi}^{\neg\Psi}(\tau). \tag{8}$$

Algorithm 2 shows the generic procedure for traversing the state evolution tree. It starts with the root of the tree, u_0, i.e., the node with the initial state of the system, property Ψ, time of interest τ, and the probability threshold q. During the exploration process we try to expand the most

promising node at each time, hence we try to use a *best-first search* policy. For this, we assign to each node a score indicating its importance, and expand nodes in the order of their scores. As we address later, this value can depend on the probability of being in that node, its associated state, or the formula under investigation. We keep the nodes that have been generated but not yet expanded in set Q, stored as a sorted data structure based on the node scores, e.g., a priority queue. At the beginning, both $\hat{\pi}^{\Psi}(\tau)$ and $\hat{\pi}^{\neg\Psi}(\tau)$ are initialized to zero since no node has been visited yet, and Q contains only u_0 (lines 1-2). While Q is not empty or the threshold conditions for Ψ holding are not met, the algorithm continues (line 3). In line 4, the node with the highest score is chosen by calling $Q.pop()$ and expanded in line 5, based on Algorithm 1. If any of the new nodes is a goal node, i.e., its entry time point is equal to τ, it contributes to either $\hat{\pi}^{\Psi}(\tau)$ or $\hat{\pi}^{\neg\Psi}(\tau)$, depending on whether its marking satisfies Ψ or not (lines 7-9). Otherwise, if the new node is not a goal node, it is added to Q, so that it can be expanded later (line 11).

Algorithm 2 $Traverse(u_0, \Psi, \tau, q)$

Require: Tree root, u_0, property Ψ, time of interest τ and probability threshold q.

```
1: π̂^Ψ(τ) ← 0, π̂^¬Ψ(τ) ← 0
2: Q ← u_0
3: while Q ≠ ∅ or π̂^Ψ(τ) ≥ q or π̂^¬Ψ(τ) > 1 − q do
4:     u ← Q.pop()
5:     U ← expand(u, τ)
6:     for all u_i ∈ U do
7:         if time(u_i) = τ then
8:             π̂^Ψ(τ) = π̂^Ψ(τ) + π^Ψ_{u_i}(τ)
9:             π̂^¬Ψ(τ) = π̂^¬Ψ(τ) + π^¬Ψ_{u_i}(τ)
10:        else
11:            Q.insert(u_i)
```

5.3 Expansion policies

As mentioned earlier, we use a best-first search [16] to traverse the state evolution tree, therefore it is important to assign a score to each visited node. This section discusses different scoring policies. In the most general form, as is represented in the A^* search method [10], each node u is assigned a score function, $f(u) = g(u) + h(u)$, where $g(u)$ is the actual gain of reaching node u and $h(u)$ is a heuristic function estimating the future gain of reaching a goal node from u. Therefore, $f(u)$ is an estimation for the score of reaching a goal node through node u.

The algorithm introduced in the previous section accumulates the probability that Ψ holds (or does not hold) at each iteration, and we are interested in reaching a probability threshold as soon as possible. Hence, nodes that contribute more probability mass to either $\hat{\pi}^{\Psi}(\tau)$ or $\hat{\pi}^{\neg\Psi}(\tau)$, should to be chosen earlier than others. The most obvious choice would be $g(u) = p(u)$, as given in Equation (3), and $h(u) = 0$. This, resembles a *greedy* approach [4], since only our current knowledge of the state evolution tree is used to choose the next node. However, if the probability distribution of nodes follows (or is close to) a uniform distribution this is no different from choosing nodes randomly, as we are not imposing any preference over nodes. On the other hand, choosing a node that leads to a goal node sooner may also be preferable since it reduces the number of intermediate nodes. This implies that nodes with lower distance to the time of interest $t = \tau$ are also advantageous. Moreover, depending on the given formula Ψ and the probability threshold q, one may choose $h(u)$ with the goal of steering the search procedure to the portion of the state space that leads to accumulating $\hat{\pi}^{\Psi}(\tau)$ or $\hat{\pi}^{\neg\Psi}(\tau)$ more effectively.

Based on these considerations, one may choose different gain and heuristic functions depending on the structure of the state evolution tree, the probability distribution of nodes, and the given property to investigate. We empirically discuss these ideas further in the next section.

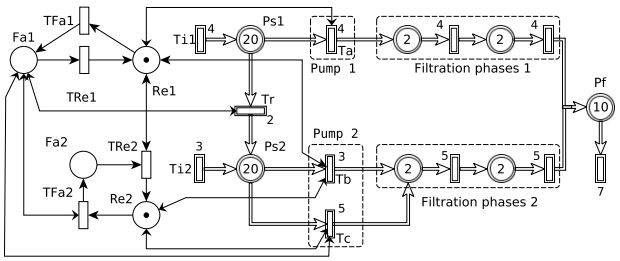

Figure 3: A Petri net model for a water treatment facility with possible cascading failure.

6. CASE STUDY

This section shows the feasibility of the proposed approach. Figure 3 shows a Petri net model for a water treatment facility, a modified version of the one given in [7]. This model consists of two production lines providing the content of the final storage, represented by the continuous place Pf, from which water is distributed to customers with a constant rate of 7. Each production line starts with a large water softening tank (Ps_1 and Ps_2). These two tanks are followed by different filtration phases to remove contaminants from the input water. Each filtration phase is modelled by a continuous transition and place. The total water input to each of these lines is modelled by continuous transitions Ti_1 and Ti_2, with rates 4 and 3, respectively. Therefore, under normal operation, the total intake of the system matches the total requests.

Continuous transition Ta models pump 1, responsible for transporting water from the softening tank to the filtration tanks in the first production line. In the second production line, continuous transitions Tb and Tc together model pump 2, with the same role as in the first production line. This is because this pump may work with different rates depending on the conditions of the entire system. These pumps connecting the softening tanks to the rest of the phases are the vulnerable points of the system, and the aim of this analysis is to investigate the resilience of the system when there are failures in these pumps.

PTT TFa_1 models the probabilistic failure of pump 1, i.e, transition Ta. If a failure occurs in the first production line, a portion of its load is forwarded to the second production line. This is done through transition Tr with rate 2, and causes extra load for the second production line. At the time of Ta's failure, transition Tb, which was working with rate 3, is disabled and instead transition Tc, with rate 5, is enabled. This is simulating the fact that pump 2 is working under extra load. This extra load may, however, result in a

cascaded failure in pump 2, modelled by PTT TFa_2. These two failures are also accompanied by probabilistic repairs, modelled by PTTs TRe_1 and TRe_2, respectively.

This model contains four PTTs, which may fire several times, by being enabled and disabled consecutively. Since these PTTs model failure and repair, one may expect that the firing probability distribution is increasing, i.e., the probability of firing grows for larger firing times. To model this we use a linear increase scheme, i.e., we consider a discretized version of a uniform cumulative distribution for their probability density function. In other words, we assign each PTT an interval $[0, T_m]$ and an integer n, where T_m is the latest time at which the PTT will fire, and n is the number of equally distanced discretization points, i.e., the number of possible firing times.

In this model, for PTTs TFa_1, TRe_1, TFa_2, and TRe_2, we assume maximum firing times of 12, 6, 15, and 6, which for large n results in average firing times of, 8, 4, 10, and 4, respectively. With this setting, there will be a failure and repair for sure, at most at the respective maximum times. It is possible to assign each PTT with different probability distributions, but we have considered the same distributions for all PTTs in this model, for simplicity.

During the evolution of the system, at most two PTTs are enabled simultaneously. So, assuming n possible firing times, the maximum branching factor is $b_{max} = 2n$. Among the four PTTs present in the system, the minimum time difference between two possible firings is $\delta = \min\{12, 6, 15\}/n = 6/n$. Hence, for the time of interest τ, the total number of nodes is at most equal to:

$$N = (2n)^{\frac{n\tau}{6}}. \tag{9}$$

We investigate the probability of not having an empty final storage:

$$\Psi_1 = x_{Pf} > 0.$$

For the first scenario, as discussed in Section 5.3, we favour nodes contributing higher probability, hence we set $f_1(u) = p(u)$. We can think of $p(u)$ as the total information we have so far, i.e., $g(u)$, which is our gain by choosing u. Hence, this score function implies that the heuristic function $h_1(u)$ is zero. Table 1 shows the scalability of the approach for different values of n, the number of firing times of the PTTs. The estimation of the state space size is computed based on (9), for the time of interest $\tau = 24$. The unassigned probability is defined as $\pi_{un}(24) = 1-(\hat{\pi}^{\Psi_1}(24)+\hat{\pi}^{\neg\Psi_1}(24))$; this is the probability that still is available for the portion of state space that we have not seen, therefore, we do not know how it contributes to either Ψ_1 or $\neg\Psi_1$. Moreover, the range providing upper and lower approximation for $\pi^{\Psi_1}(24)$ is based on (7).

Table 1: Approximated probability of Ψ_1 holding at time $\tau = 24$, for different values of n (rounded to 3 decimal places).

n	N (9)	# seen nodes	$\hat{\pi}^{\Psi_1}(24)$	$\hat{\pi}^{\neg\Psi_1}(24)$	$\pi_{un}(24)$	$\pi^{\Psi_1}(24)$ range
5	10^{20}	1.5×10^5	0.841	0.158	10^{-5}	$[0.841, 0.842]$
10	10^{40}	7.0×10^6	0.809	0.189	10^{-3}	$[0.809, 0.812]$
30	10^{120}	1.2×10^7	0.692	0.137	0.170	$[0.692, 0.863]$

As can be seen in Table 1, for $n = 30$, by generating around 12 million nodes (10^{-111}% of the total state space), we face a 17% unassigned. According to inequality (8), for the case of $n = 30$, we cannot reach a conclusion if the given threshold q satisfies: $0.692 \leq q < 0.863$. Even though the computed range in the last row of Table 1 is large, we are able to reach a conclusion, especially when checking for thresholds close to zero or one. This is often the case for the verification of safety properties, which either express to have a very low probability of service failure, or a high probability of recovery from a failure. Indeed, the proposed algorithm is aimed toward this case. More specifically, the final range is "pushed away" from 0 and 1, by accumulating both the probability of the given property and of its negation.

Next, we examine the effectiveness of the heuristics. As mentioned in the previous section, this can be done by considering the structure of the model and the given property. We choose a heuristic which favours nodes more likely to lead to a state with fewer failed pumps. For this, we can count the number operational pumps, so we propose $h_2(u) = p(u).(n_{Re_1} + n_{Re_2})/2$. This means that nodes that contribute higher probability and less potential for leading to states where the property Ψ_1 is violated are chosen first. We do not change $g(u)$ from the previous case, so we have $f_2(u) = p(u) + h_2(u)$.

Figure 4 shows the results for the proposed heuristic. The vertical axis represents the number of generated nodes, and the horizontal axis is the probability for different aspects of the property Ψ_1. The solid and dashed lines are the values of $\hat{\pi}^{\Psi_1}(24)$, $1 - \hat{\pi}^{\neg\Psi_1}(24)$, and $\pi_{un}(24)$, for f_1 and f_2, i.e., the score functions with or without heuristic functions, respectively. The shaded area depicts the upper and lower approximation of $\pi^{\Psi_1}(24)$, which becomes narrower as more nodes are generated. One can see that the heuristic function embodied in f_2, which guides the process towards nodes with less potential of violating Ψ_1, results in a slightly faster accumulation of $\hat{\pi}^{\Psi_1}(24)$ (green, solid), hence, less unassigned probability, $\pi_{un}(24)$ (red, solid).

One may notice that in the beginning the probability mass is accumulated faster for both f_1 and f_2. This is because, as we first choose nodes contributing high probabilities, after a while the unassigned probability $\pi_{un}(\tau)$ is distributed among many nodes, each contributing little probability mass. So, the algorithm must traverse many more nodes than at the beginning of the process to accumulate the remaining probability.

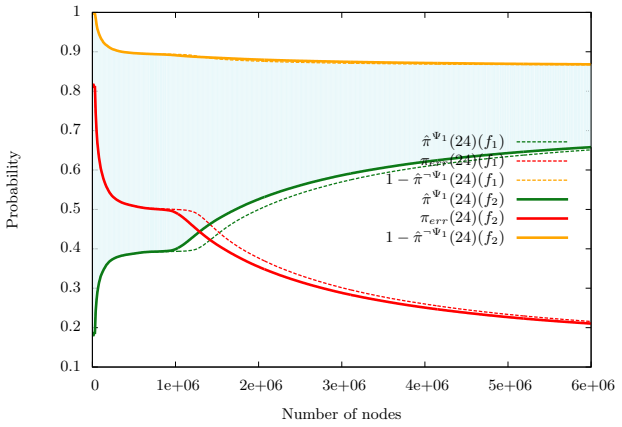

Figure 4: Illustration of performance of different score functions, f_1 and f_2 for accumulating $\hat{\pi}^{\Psi_i}(24)$ and $\hat{\pi}^{\neg\Psi_i}(24)$, $i \in \{1, 2\}$, for up to 12 million nodes.

7. CONCLUSION

This work extends the modelling formalism of hybrid Petri nets with Probabilistic Timed Transitions (PTTs), which can model discrete probabilistic jumps. We introduced and analysed the state evolution tree, and explained how effective search strategies can be used to compute upper and lower approximations of the probability that a given property holds at a particular point in time. We have shown how the exploration of the state evolution tree can be guided by heuristics towards nodes that contribute more probability mass to the properties of interest, thereby achieving better approximation bounds faster. We have shown that, even for models with many PTTs, we are able to verify whether the probability that a certain property holds exceeds a given threshold. Unlike many other works on probabilistic hybrid automata, we focused on the applicability of the method by tailoring it toward the validation of safety properties for large systems. In other words, as we showed by means of a realistic case study, our intelligent accumulation of probabilities can reach a conclusion for cases where the given probability threshold is close to 0 or 1.

Future work may investigate the possibility of using (edge-valued) decision diagrams for a more efficient state representation, similar to what has been done for integer timed Petri nets [19]. Furthermore, for stochastic hybrid systems, one could combine the current work with region-based state space representations as in [7, 8]. Each node will then be associated with both a region and the corresponding marking, which will result in an even higher branching factor.

Acknowledgement

This work was supported by CTIT for funding travels to Iowa State University, USA. Anne Remke was funded by an NWO Veni grant. Gianfranco Ciardo was supported in part by the National Science Foundation under grant CCF-1442586.

References

[1] A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724 – 2734, 2008.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, S. Yovine, P.H. Ho, X. Nicollin, A. Olivero, and J. Sifakis. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138:3–34, 1995.

[3] A. Bell. *Distributed evaluation of stochastic Petri nets*. PhD thesis, RWTH Aachen University, 2004.

[4] T. H. Cormen, C. E. Leiserso, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 1990.

[5] R. David and H. Alla. On hybrid Petri nets. *Discrete Event Dynamic Systems*, 3:9–40, 2001.

[6] R. David and H. Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer, 2010.

[7] H. Ghasemieh, A. Remke, and B. R. Haverkort. Hybrid Petri nets with multiple stochastic transition firings. In *8th Int. Conf. on Performance Evaluation Methodologies and Tools*. 2014.

[8] H. Ghasemieh, A. Remke, B.R. Haverkort, and M Gribaudo. Region-Based Analysis of Hybrid Petri Nets with a Single General One-Shot Transition. In *Formal Modeling and Analysis of Timed Systems*, volume 7595 of *LNCS*, pages 139–154. Springer, 2012.

[9] M. Gribaudo and A. Remke. Hybrid Petri Nets with General One-Shot Transitions for Dependability Evaluation of Fluid Critical Infrastructures. In *12th Int. Symp. on High Assurance Systems Engineering*, pages 84–93. IEEE, 2010.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[11] G. Horton, V.G. Kulkarni, D.M. Nicol, and K.S. Trivedi. Fluid stochastic Petri nets: Theory, applications, and solution techniques. *European Journal of Operational Research*, 105(1):184–201, 1998.

[12] D. Klink, A. Remke, B. R. Haverkort, and J. Katoen. Time-Bounded Reachability in Tree-Structured QBDs by Abstraction. In *6th Int. Conf. on the Quantitative Evaluation of Systems*, pages 133–142. IEEE Computer Society, 2009.

[13] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

[14] M.K. Molloy. Discrete time stochastic petri nets. *IEEE Transactions on Software Engineering*, SE-11(4):417–423, 1985.

[15] A.P.A. Van Moorsel and B.R. Haverkort. Probabilistic evaluation for the analytical solution of large Markov models: Algorithms and tool support . *Microelectronics Reliability*, 36(6):733 – 755, 1996.

[16] J. Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[17] A. Remke, B.R. Haverkort, and L. Cloth. CSL model checking algorithms for QBDs. *Theoretical Computer Science*, 382(1):24 – 41, 2007.

[18] J. Sproston. Decidable Model Checking of Probabilistic Hybrid Automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 1926 of *LNCS*, pages 31–45. Springer, 2000.

[19] M. Wan and G. Ciardo. Symbolic Reachability Analysis of Integer Timed Petri Nets. In *SOFSEM 2009: Theory and Practice of Computer Science*, volume 5404 of *LNCS*, pages 595–608. Springer, 2009.

[20] L. Zhang, Z. She, S. Ratschan, H. Hermanns, and E. Hahn. Safety verification for probabilistic hybrid systems. In *Computer Aided Verification*, volume 6174 of *LNCS*, pages 196–211. Springer, 2010.