Cost-optimal timed trace synthesis for scheduling of intermittent embedded systems

Antoine Bernabeu, Mikaël Briday, Sébastien Faucou, Jean-Luc Béchennec, Olivier H. Roux

Cost-optimal timed trace synthesis for scheduling of intermittent embedded systems

Antoine Bernabeu[1*], Jean-Luc Béchennec[3†], Mikael Briday[1†], Sebastien Faucou[2†] and Olivier H. Roux[1†]

[1]École Centrale Nantes, LS2N, UMR 6004, F-44000 Nantes, France.
[2]Nantes Université, LS2N, UMR 6004, F-44000 Nantes, France.
[3]CNRS, LS2N, UMR 6004, F-44000 Nantes, France.

*Corresponding author(s). E-mail(s): antoine.bernabeu@ls2n.fr; Contributing authors: jean-luc.bechennec@ls2n.fr; mikael.briday@ls2n.fr; sebastien.faucou@ls2n.fr; olivier-h.roux@ls2n.fr;
†These authors contributed equally to this work.

Abstract

Intermittent computing is an emerging paradigm for systems without batteries and powered by intermittent energy sources. This paradigm promises a more energy-efficient design of computing systems. It seems particularly well suited to the field of connected sensors that form the first level of the Internet of Things. This application domain requires a reactive computing model. The definition of an intermittent and reactive model is a problem that has not yet been fully explored in the literature. In this paper, we focus on the modeling and analysis of intermittent reactive systems. We first introduce an extension of Time Petri Nets with cost to model the different dimensions of the system: concurrency, real time, energy consumption and reward representing the gains generated by the system when it has succeeded in carrying out certain actions. We then aim to synthesize optimal runs of the model that achieve the highest possible reward under a given cost (energy) constraint. We propose a symbolic algorithm for constrained-cost state space computation and prove its termination. We then present algorithms for the synthesis of the optimal traces from an exhaustive or partial state space exploration. We finally illustrate the cost-optimal traces synthesis on a case study and show how that can be used online for joint management of computing time and energy.

Keywords: Petri net, Intermittent computing, Optimization, Scheduling

1 Introduction

Today, several arguments push towards a battery-less design for autonomous smart sensors: batteries have a limited number of cycles thus requiring maintenance actions that may be costly and/or difficult to carry out, and batteries are a source of pollution during the production, recycling and disposal phases of systems.

At the same time, there is a clear trend to move computation closer to data sources, including smart sensors. Near-sensor computing enables low-latency functions that are not feasible when a round-trip to the cloud is required. Near-sensor computing helps limit the amount of data that needs to be sent (by filtering out noise and/or performing transformation steps): since computation is much more energy efficient than communication, trading communication against computation usually improves the overall energy efficiency. Moreover, limiting the broadcast of raw data remains a good practice with respect to privacy.

Is it possible to eliminate the battery while increasing the complexity of the calculations performed by the smart sensors? One possible response to this question is given by the emergence of intermittent computing, a model of computation for systems powered by intermittent sources, and its application for battery-less autonomous smart sensors [1, 2]. It is enabled by the availability of ambient energy harvesting devices and ultra-low-power micro-controllers built around efficient non-volatile memory. In an intermittent system, power loss is a normal event that does not hinder forward progress of computations. To do so, the system integrates mechanisms to save progress when power goes down, and resume to execution when power is back. Thus, a long computation can be spread over several cycles.

Our contribution

In this article, we study the problem of modeling and analyzing concurrent intermittent embedded systems. For the modeling part, we introduce an extension of *Time Petri Nets* (TPN for short), with linear cost and reward. The cost is used to model the energy consumption of the system as a function of its current activities, which in turn are modeled by the state of the underlying TPN. The reward makes it possible to model the gains generated by the system when it has succeeded in carrying out certain actions (*e.g.*, finishing a computation, or sending a message). In terms of analysis, we are interested in the synthesis of optimal traces, *i.e.*, finding runs in the model that achieve the highest possible reward under a given cost (energy) constraint. We propose a symbolic algorithm for constrained-cost state space computation and prove its termination. We then present two exact algorithms and an approximate one allowing to synthesize the optimal traces from an exhaustive or partial state space exploration. We also discuss how the traces computed by this analysis provide relevant information for designers, more specifically for scheduling the different activities, including checkpointing, on a system according to local constraints (*e.g.*, remaining energy, or available memory).

Outline

The article is organized according to the following plan: in section 2 we present some background on intermittent systems and formal models with time and costs; in section 3 we describe more precisely the type of system we are interested in; in section 4 we introduce the syntax and semantics of the modeling formalism we use: cost time Petri nets; in section 5 we formalize the problem we aim to solve and give an exact algorithm based on an exhaustive state space exploration; in section 6, we describe two heuristics to guide the exploration of the state space: one of them towards an optimal solution and the other towards an approximate one; finally, in section 7, we present a case study including the modeling and analysis of an intermittent smart sensor, and discuss how the results produced by the analysis can help in the runtime scheduling of such a system.

2 Related works

2.1 Models of computation for intermittent computing

From a software perspective, the main challenge is to define and support an intermittent computing model, *i.e.*, one in which power losses are a normal event that does not hinder forward progress. To do so, the system integrates mechanisms to save its state when the power goes out and to restore it when the power comes back on. Thus, a long computation can be spread over several power loss and recovery cycles. A state of the system saved in non-volatile memory (NVM) is called a checkpoint, and by extension the name is also used to designate the action to build the checkpoint.

In the literature dealing with intermittent computing, early work focused on dynamic insertion of checkpoints through a dedicated runtime [1, 3]. Then, an alternative direction has been explored in which the program is decomposed into subsequence, usually named tasks, executed according to a transactionnal semantics where the commit part correspond to a checkpoint. The tasks are constructed either by the programmer [4] or by the compiler [5, 6], possibly with an adaptative phase during execution to activate or deactivate checkpoints. Task-based intermittent computing can be with rollbacks [4] if it runs until a power loss, or without if it stops the system after a checkpoint if it cannot safely reach the next checkpoint with the available energy [6]. Task-based solution facilitates static reasoning about the behavior of the system, but may induce inefficient use of energy (with rollback) or raise the issue of estimating the available energy (without rollback).

Concerning the programming model, much work assumes that the program is sequential, can always be delayed, interrupted, or replayed several times. Some recent work extend the model to integrate mechanisms from concurrent and reactive computing. Sytare [7] is a software layer that handles asynchronous device and device drivers for intermittent systems. Coati [8] also allows the integration of interrupt routines. InK [9] is similar to an event-driven kernel for intermittent systems. Catnap [10] adopts a similar event-driven model and adds the possibility of reserving a part of the energy buffer to provide a minimum notion of quality of service.

In this work, we are interested in an intermittent computing model with concurrent threads and static checkpoints without rollback.

2.2 Modeling intermittent systems

If one is primarily interested in formal models, the work in the state of the art focuses on the functional aspects of intermittent systems with the objective of establishing proofs of correctness [11, 12]. Physical resources (*e.g.*, computation time, or energy) are modeled in a very abstract way.

Outside the framework of formal models, several authors propose solutions to simulate the execution of an intermittent system, with a particular interest in modeling energy consumption. Some of them are oriented towards the static prediction of the worst case of energy consumption [13] while others, such as the *framework* Fused [14], are oriented towards simulation. Simulation models include lots of details but have a significant complexity, making them unsuitable for design space exploration.

To enable the development of higher-level models, power measurement platforms have been developed [15, 16]. These platforms are mainly based on current measurement in order to measure the energy consumption of the system and its different modules. They assume that the supply voltage is constant, which requires the integration of a voltage regulator between the energy harvesting devices and the micro-controller and external peripherals.

Finally, some work focus on the prediction of the energy produced by the harvesting devices, mainly in the case of solar panels. EWMA [17] is certainly the most widely used algorithm because of its simplicity, but unfortunately its prediction errors rate is important. Pro-Energy-VLT [18], based on embedded profiles, is much more accurate but requires memory space to be used online so it does not fit all use-cases, especially in the domain of smart sensors.

In this work, we propose a high-level modeling approach for intermittent system with quantitative representation both for execution time and energy based on TPN. The embedded energy model is based on measurements (see section 7.1) at the platform level: it is not limited to the activities of the CPU. It is close to the work of Berthou *et al.* [16] but we do not assume the integration of a regulator so the supply voltage is not constant.

2.3 Formal models with time and costs

Priced or cost timed models are suitable models for real-time systems, when we have to take into account that the behavior of the system is constrained by the consumption of different types of resources and/or the accumulation of some cost during its execution. These models allow to define the accumulated cost for a given run of the system. They have been first introduced in [19] for Priced Timed Automata (PTA) and in [20] for Priced Timed arc Petri net (PTPN).

The problem of determining the minimum cost for reaching a designated set of target states is particularly interesting. This problem has been proved decidable in [19] for PTA with non-negative integer costs. A solution based on a forward exploration of zones extended with linear cost functions has been proposed in [21, 22] for PTA and in [20] for PTPN. Algorithms to solve the min-cost problem have been implemented in modeling and verification toolboxes: UPPAAL CORA for PTA [23], and Roméo for TPN [24].

In [25], the authors consider cost time Petri nets where each transition has a firing cost and each marking has a rate cost. They solve the optimal-cost reachability problem by revisiting the state class graph method [26] to include costs.

Another interesting problem is the existential lower-bound constrained problem. Given an initial credit (weight), the existential lower-bound constrained problem is to decide whether there exists an infinite run where the cost is continuously maintained non-negative. In [27], the authors show that this problem is undecidable for PTA but become decidable when only bounded-duration runs are considered.

3 System overview

3.1 System architecture

A battery-less autonomous smart sensor is powered by energy harvested from its environment. It incorporates a super-capacitor used as an energy buffer, which keeps the system operating for a few moments when insufficient energy is harvested. To implement an intermittent computing model, these few moments must be used to progress as much as possible in the on-going computations and to save the progress, *i.e.*, to save the volatile state of the system to a non-volatile memory (NVM for short). To enable this last step, the system must integrate a sufficiently powerful and efficient NVM. One example is ferroelectric RAM, or FRAM, which offers better performance, energy efficiency, and lifetime than Flash memory [28]. FRAM memory have been integrated for instance by Texas Instrument as a drop-in replacement of NOR-Flash, to serve as NVM a series of micro-controllers built around the MSP430 16-bit CPU. Systems of this serie are typically a good fit for autonomous smart sensor running an intermittent computing model and correspond to the type of system we are focusing on in this paper. More precisely, we consider a system with the following hardware features:

- A platform offering non-volatile memory with low power operation.
- A limited energy storage capability using a super-capacitor. Typically, with a full charge of the super-capacitor and not considering energy harvesting, the system powering time ranges from tens of seconds to minutes depending on the devices that are powered for a given task.
- An energy harvesting device: photo-voltaic, wind, thermo-electric, ... The harvested energy is stored in the super-capacitor and used in parallel to power the system. The harvest, depending on the circumstances, can be sufficient to feed the system whatever its consumption, or insufficient, or even nil. In the two latter cases, once the super-capacitor charge is too low, the system must go through voluntary stops to allow the recharging at a sufficient level.
- And a direct supply of the system without any voltage regulator.

This last item is of paramount importance when one gets interested in modeling the energy consumption of such a system. The energy supplied by the harvesting device is stored in the super-capacitor. This energy E, is a function of the voltage according to equation (1), where C is the capacity of the super-capacitor and V_{cc} is the voltage across the super-capacitor.

$$E = \frac{1}{2} \times C \times V_{cc}^2 \tag{1}$$

The power P consumed by the system is the sum of the static power P_{stat}, almost constant and the dynamic power P_{dyn} which varies in function of the activity of the system : $P = P_{stat} + P_{dyn}$. On the type of system considered (CMOS technology with low frequency, from few MHz to few dozen MHz), the static power is very low compared to dynamic power. The static power consumption can be estimated by measuring the energy consumption while the intermittent system is in low energy consumption with all the clocks stopped but the CPU still supplied.

The dynamic power consumed by the CMOS circuit is ruled by equation (2), where f_{clk} is the frequency of the system (in our case the micro-controller), C_L is the sum of capacities of capacitor charged and discharged during operations, and V_{dd} is the supplied voltage.

$$P_{dyn} = f_{clk} \times C_L \times V_{dd}^2 \tag{2}$$

According to equation (2), dynamic power consumption depends, like the energy, on the square of the supply voltage. As we consider a system fed directly from the super-capacitor (*i.e.*, without the intermediary of a voltage regulator that would keep the supply voltage constant) the supply voltage equals the super-capacitor voltage: $V_{cc} = V_{dd}$. The consumption is non-linear and follows a law in V^2. The result is that the voltage varies linearly with time as the energy buffer depletes. Thus, the available energy is given by the supply voltage and the energy consumption model reduces to a linear model (the voltage drop as a function of time).

Notice that term C_L used in equation (2) depends on the activities on-going at system level and is not constant accross time. Thus, the slope of the linear model is not constant either, it changes slightly each time a transistor switches in the system. Thus, in this framework, the energy consumption model of a system can be more or less accurate, depending on the level of detail that is given to identify the different slopes. One could probably distinguish one by one each the execution of CPU instruction, or even the different execution stages of a given instruction. This level of detail is neither useful nor desirable for the type of problem that interests us. To better understand the level of detail relevant to our case, we refer the reader to the section 7.1 which presents the model we have built for a case study. This model was obtained from a set of measurements performed on a real target.

We have chosen not to include a buck/boost regulator in this study in order not to be penalized by its efficiency. However, if there is one, we should rely on the input current of the system to keep a linear evolution, as in [16].

3.2 Model of computation

We consider a concurrent, task-based[1], intermittent model of computation. In this model, a workload is composed of different tasks. These tasks can be software, hardware, or a mix of both[2]. The presence of hardware tasks implies a parallel execution model: a hardware task can execute at the same time as another task (hardware or software or mixed) that does not use the same resources.

As usual in intermittent computing models, a task has a transactional semantics: once started, either it ends before the next power loss, or the progress is lost. Although this constraint could be relaxed for software tasks (by including their volatile state in the checkpoint), it is more difficult for software/hardware and for hardware ones that use hardware devices whose context is inaccessible (*e.g.*, a computing accelerator) or those using a device that cannot be interrupted (*e.g.*, a radio transmitter when sending a frame).

Tasks can be linked by precedence relations. For example, a set of data can be sampled by a sensor, then processed by software functions, and finally the result is sent to a server by radio transmission. Intermediate buffers can be inserted between tasks to allow for a pipeline execution model. For example, several successive processing results can be stored in memory while transmission is deferred until the available energy is sufficient to allow it.

The interest of the pipelined execution model is to offer a greater flexibility to the scheduler, including in the presence of precedence constraints, to choose the next tasks to execute according to the available resources (memory, energy)

[1] The term task is used here in the sense given to it in the literature on intermittent computing, and not in the sense given to it in the literature on system scheduling.

[2] For example, the capture of analog sensor data at regular intervals can be done without software involvement by using a timer, the Analog to Digital Converter device (ADC) and the Direct Memory Access device (DMA).

and the functional objectives (forward progress). Moreover, it is a general model, which includes the non-pipelined case (obtained by considering buffers of size 0 between the different "stages" of the pipeline).

It is important to note that even in the presence of precedence constraints linking all the tasks of a system, the pipelined execution model makes it possible to partially exploit the parallelism offered by the platform[3].

In this article, we focus on modeling and analysis of systems that implement this model of computation. We leave for future work the design of an actual implementation. However, we can emphasize that the model of computation uses classical mechanisms, whose implementation has already been studied in the literature [8–10].

3.3 First formulation of problem

Let an intermittent smart sensor using the architecture described in section 3.1 and implementing the model of computation described in section 3.2. We are interested in the problem of scheduling the tasks of the system to maximize its energy efficiency. For this, we have two objectives:

- on the one hand, when choices have to be made between several tasks, energy should be consumed to execute those that most advance the system toward its functional goals;
- on the other hand, energy should never be used to execute a task that cannot be guaranteed to be completed before the next power loss.

The formulation of this problem reveals several elements. First, an energy consumption must be attached to a task, and more broadly, we must be able to compute the instantaneous energy consumption of the system at a given instant as a function of the on-going tasks. As previously explained, a linear time model can be used. Each task is then associated with a slope that represents its consumption. The system consumption is obtained by combining the slopes of the current tasks. We describe in section 7.1 how we obtained such a model for a real system.

Second, there must be a quantification of the way a task makes the system progress towards its functional objectives. Of course this is not an objective physical quantity. It is rather a value that the system designer must be able to associate with each task. We denote *reward* the quantification of these functional objectives. This idea is not new and value-based scheduling has already been explored in different application domains [29–31]. To the best of our knowledge, this is the first time it has been used in conjunction with an intermittent computing model.

Given a model that captures all the dimensions listed above, the problem we wish to solve is to obtain the trace of an execution leading to a state where the *reward* is maximum among all states that are reachable under a given energy constraint. The energy constraint represents the maximum energy that can be

[3]The pipelined execution model remains general, as we can still consider a software task as a whole, with a single-stage pipeline.

stored in the super-capacitor. If more than one state can gain the maximum *reward*, we are interested in finding which of them can be reached with the least amount of energy. We explain in the following sections how to formalize and solve this problem using an extension of Time Petri nets with cost. Then, we illustrate with a case study how the solution of this problem can be used to efficiently schedule an intermittent system.

4 Cost Time Petri Nets

The theory of *Petri Nets* provides a general framework to specify the behavior of concurrent reactive systems. *Time Petri Nets* (TPN), introduced in [32] to take into account real-time specifications, extend Petri nets with time constraints on transition firings. Efficient reachability analysis methods, usually based on state space abstraction such as the state class graph, allow to represent firing sequences and reachable markings.

4.1 Preliminaries

We denote $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$ respectively the set of integers, natural, rational and real numbers. We consider 0 to be an element of $\mathbb{N}$ and let $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$. For $n \in \mathbb{N}$, we let $[\![0, n]\!]$ denote the set $\{i \in \mathbb{N} \mid i \leq n\}$. The set of non empty real intervals that have rational (respectively natural numbers) or infinite endpoints is denoted $\mathcal{I}_{\mathbb{Q}_{\geq 0}}$ (respectively $\mathcal{I}_{\mathbb{N}}$). For $I \in \mathcal{I}_{\mathbb{Q}_{\geq 0}}$, $\underline{I}$ denotes its left end-point and $\overline{I}$ denotes its right end-point if I is bounded and ∞ otherwise. Moreover, for any $d \in \mathbb{R}_{\geq 0}$, we let $I \dot{-} d$ be the interval defined by $\{\theta - d \mid \theta \in I \wedge \theta - d \geq 0\}$.

For F and F', two systems of linear inequalities over a set of variables X, we denote $F \equiv F'$ when they have equal solution sets over X. Moreover we denote by $F_{|Y}$ (with $Y \subseteq X$) the projection of F over Y (we use Fourier–Motzkin elimination of the variables Z s.t. $Y \cup Z = X$ and $Y \cap Z = \emptyset$).

4.2 Time Petri Nets

We first recall the well-known definitions of Time Petri Nets and state class graph of [26, 33].

Definition 1 (Time Petri Net (TPN)) A *Time Petri Net* is a sextuple $\mathcal{N} = (P, T, {}^\bullet., .^\bullet, m_0, I_s)$ where

- P is a finite non-empty set of *places*,
- T is a finite set of *transitions* such that $T \cap P = \emptyset$,
- ${}^\bullet. : T \to \mathbb{N}^P$ is the backward incidence mapping,
- $.^\bullet : T \to \mathbb{N}^P$ is the forward incidence mapping,
- $m_0 : P \to \mathbb{N}$ is the initial marking, and
- $I_s : T \to \mathcal{I}_{\mathbb{N}}$ is a function assigning a *firing interval* to each transition.

A marking is a mapping from P to $\mathbb{N}$. For a marking $m \in \mathbb{N}^P$, $m(p)$ denotes the number of tokens in place p. A Petri net $\mathcal{N}$ is said to be k-bounded or simply bounded if the number of tokens in each place does not exceed a finite number k for any marking reachable from m_0.

- A transition $t \in T$ is said to be *enabled* by a given marking $m \in \mathbb{N}^P$ if m supplies t with at least as many tokens as required by the backward incidence mapping $^\bullet$. We define the set $En(m)$ of transitions that are enabled by the marking m as $En(m) = \{t \in T \mid m \geq {}^\bullet(t)\}$
- A transition $t' \in T$ is said to be *newly enabled* by the firing of a transition t from a given marking $m \in \mathbb{N}^P$ if it is enabled by $m - {}^\bullet t + t^\bullet$ but not by $m - {}^\bullet t$. The set of transitions that are newly enabled by the firing of t from the marking m is $\mathcal{N}ewlyEn(m,t) = \big\{t' \in En(m - {}^\bullet t + t^\bullet) \mid t' \notin En(m - {}^\bullet t) \text{ or } t = t'\big\}$

Definition 2 (State) A *state* of the net $\mathcal{N}$ is a pair (m, I) in $\mathbb{N}^P \times \mathcal{I}_{\mathbb{Q}_{\geq 0}}^T$, where m is a marking of $\mathcal{N}$ and I is a function called the interval function. $I : T \to \mathcal{I}_{\mathbb{Q}_{\geq 0}}$ associates a *temporal interval* with every transition enabled by m.

Definition 3 (Semantics of a TPN) The semantics of a TPN is a timed transition system $(Q, q_0, \to)$ where:

- $Q \subseteq \mathbb{N}^P \times \mathcal{I}_{\mathbb{Q}_{\geq 0}}^T$
- $q_0 = (m_0, I_0)$ s.t. $\forall t \in En(m_0) \;\; I_0(t) = I_s(t)$
- $\to$ consists of two types of transitions:
  - discrete transitions: $(m, I) \xrightarrow{t} (m', I')$ iff
    - $m \geq {}^\bullet t$, $m' = m - {}^\bullet t + t^\bullet$ and $\underline{I(t)} = 0$,
    - $\forall t' \in En(m')$
      - $I'(t') = I_s(t')$ if $t' \in \mathcal{N}ewlyEn(m,t)$,
      - $I'(t') = I(t')$ otherwise
  - time transitions: $(m, I) \xrightarrow{\delta \in \mathbb{Q}_{\geq 0}} (m, I \dot{\dot{-}} d)$ iff $\forall t \in En(m)$, $\overline{(I \dot{\dot{-}} \delta)(t)} \geq 0$.

A run of a time Petri Net $\mathcal{N}$ is a (finite or infinite) path in its semantics starting in q_0.

We denote $(m, I) \xrightarrow{t@\delta} (m', I')$ for the sequence of elapsing δ followed by the firing of the transition $t : (m, I) \xrightarrow{\delta} (m, I \dot{\dot{-}} \delta) \xrightarrow{t} (m', I')$.

The set of runs of a TPN is denoted by Runs

We denote *sequence*(ρ) (resp. *trace*(ρ)) the projection of the run ρ over T (resp. over $T \times \mathbb{Q}_{\geq 0}$). The sequence σ (resp. the trace τ) corresponding to the run $\rho = q_0 \xrightarrow{t_0@\delta_0} q_1 \xrightarrow{t_1@\delta_1} q_2 \xrightarrow{t_2@\delta_2} q_3$ is $\sigma = sequence(\rho) = t_0t_1t_2$ (resp. $\tau = trace(\rho) = t0@\delta_0.t_1@\delta_1.t_2@\delta_2$).

Definition 4 (Discrete state graph of a TPN) The discrete state graph (DSG) of a TPN is the structure $DSG = (S, s_0, \hookrightarrow)$ where $S \in \mathbb{N}^P \times \mathcal{I}^T_{\mathbb{Q}_{\geq 0}}$, $s_0 = (m_0, I_s)$ and $s \overset{t}{\hookrightarrow} s'$ iff $\exists \delta \in \mathbb{Q}_{\geq 0} \mid s \xrightarrow{t@\delta} s'$

Any state of the DSG is a state of the semantics of the TPN and any state of the semantics which is not in the DSG is reachable from some state of the DSG by a continuous transition.

The DSG represents a dense state space in the sense that a state of the DSG may have infinite number of successors by $\overset{t}{\hookrightarrow}$. Finitely representing dense state spaces involves grouping some sets of states.

4.2.1 State Classes

For an arbitrary sequence of transitions $\sigma = t_1 \ldots t_n \in T^*$, let C_σ be the set of all states that can be reached by the sequence σ from s_0 : $C_\sigma = \{s \in S \mid s_0 \overset{t_1}{\hookrightarrow} s_1 \cdots \overset{t_n}{\hookrightarrow} s\}$.

All the states of C_σ share the same marking and can therefore be written as a pair (m, D) where m is the common marking and D is the union of the points belonging to the set of firing intervals. D is called the *firing domain*.

We denote $\cong$, the relation satisfied by two such sets of states when they have the same marking and the same firing domain.

Definition 5 Let $C_\sigma = (m, D)$ and $C'_{\sigma'} = (m', D')$ be two sets of states, $C_\sigma \cong C_{\sigma'}$ iff $m = m'$ and $D \equiv D'$.

If $C_\sigma \cong C_{\sigma'}$, any firing schedule firable from some state in C_σ is firable from state in $C_{\sigma'}$ and conversely. The state classes of [26, 33] are the above sets C_σ considered modulo $\cong$ equivalence.

Definition 6 The state class graph (SCG) of [26, 33] is defined by the set of state classes equipped with a transition relation: $C_\sigma \xrightarrow{t} X$ iff $C_{\sigma.t} \cong X$.

Hence, the SCG computes the smallest set C of state classes w.r.t. $\cong$. The SCG is finite iff the net is bounded. Moreover, the SCG is a complete and sound state space abstraction of the TPN.

Given a state class $C = (m, D)$, a point $x = (\delta_1, \delta_2, ..., \delta_n) \in D$ is composed of the values of variables $\theta_1, \theta_2, ..., \theta_n$ that refers to the firing instants in C of transitions $t_1, t_2 ... t_n$ that are enabled by m. The firing domain may be described by linear inequations of the form $\theta_i \leq k$ or $\theta_j - \theta_i \leq k'$ where $k \in \mathbb{N}$ and $k' \in \mathbb{Z}$.

4.3 Cost Time Petri Nets

Definition 7 (Cost Time Petri Net (cTPN)) A *Cost Time Petri Net* is a tuple $\mathcal{N}_c = (P, T, {}^{\bullet}., .^{\bullet}, m_0, I_s, \omega, cr)$ where

- $\mathcal{N} = (P, T, {}^{\bullet}., .^{\bullet}, m_0, I_s)$ is a TPN.
- $\omega : T \to \mathbb{N}$ is the discrete reward function.
- $cr : \mathbb{N}^P \to \mathbb{Z}$ is the cost rate function. cr is a linear function over markings with integer coefficients.

Definition 8 (Semantics of a cTPN) The semantics of a cTPN $\mathcal{N}_c = (P, T, {}^{\bullet}., .^{\bullet}, m_0, I_s, \omega, cr)$ is the semantics of the TPN $\mathcal{N} = (P, T, {}^{\bullet}., .^{\bullet}, m_0, I_s)$.

The cost state of a cTPN is $(m, I, \mathcal{R}, \mathcal{C}) \in \mathbb{N}^P \times \mathcal{I}^T_{\mathbb{Q}_{\geq 0}} \times \mathbb{N} \times \mathbb{R}$, where (m, I) is a TPN state and $\mathcal{R}$ and $\mathcal{C}$ are respectively the accumulation from the initial state of the reward and the cost of the discrete and timed transitions of a run that leads to (m, I).

- the reward of a discrete transition $(m, I, \mathcal{R}, \mathcal{C}) \xrightarrow{t} (m', I', \mathcal{R}', \mathcal{C}')$ is $\mathcal{R}' - \mathcal{R} = \omega(t)$
- the cost of a timed transition $(m, I, \mathcal{R}, \mathcal{C}) \xrightarrow{d} (m, I', \mathcal{R}', \mathcal{C}')$ is $\mathcal{C}' - \mathcal{C} = d \times cr(m)$;

Definition 9 (Cost of a run (*cost*)) The cost of a run $\rho = (m_0, I_0, \mathcal{R}_0, \mathcal{C}_0) \xrightarrow{t_0@\delta_0} (m_1, I_1, \mathcal{R}_1, \mathcal{C}_1) \xrightarrow{t_1@\delta_1} (m_2, I_2, \mathcal{R}_2, \mathcal{C}_2) \cdots \xrightarrow{t_{n-1}@\delta_{n-1}} (m_n, I_n, \mathcal{R}_n, \mathcal{C}_n)$ is

$$cost(\rho) = \mathcal{C}_n = \sum_{i=0}^{n-1} \delta_i \times cr(m_i)$$

Definition 10 (Reward of a run) The reward of a run $\rho = (m_0, I_0, \mathcal{R}_0, \mathcal{C}_0) \xrightarrow{t_0@\delta_0} (m_1, I_1, \mathcal{R}_1, \mathcal{C}_1) \xrightarrow{t_1@\delta_1} (m_2, I_2, \mathcal{R}_2, \mathcal{C}_2) \cdots \xrightarrow{t_{n-1}@\delta_{n-1}} (m_n, I_n, \mathcal{R}_n, \mathcal{C}_n)$ is

$$reward(\rho) = \mathcal{R}_n = \sum_{i=0}^{n-1} \omega(t_i)$$

4.4 Example

Let us consider the cTPN of Figure 1. The cost rate is $cr = 2 \times p_2 + 3 \times p_3 + 3 \times p_5 + 5 \times p_6 + 5 \times p_7 + 5 \times p_8 + 5 \times p_9$ where p_i is an abbreviation for $m(p_i)$ and $m(p_i)$ is the number of tokens in p_i. The firing of the sequence t_2, t_1, t_4 and t_6 respectively at absolute dates 1.4, 2, 5 and 6 gives the following run: $\rho = q_0 \xrightarrow{t_2@1.4} q_1 \xrightarrow{t_1@0.6} q_2 \xrightarrow{t_4@3} q_3 \xrightarrow{t_6@1} q_4$ with

- $q_0 = (m_0, I_0, \mathcal{R}_0, \mathcal{C}_0) = \left(\left\{ \begin{array}{c} p1 \\ p2 \\ p3 \end{array} \right\}, \begin{array}{l} t_1 : [2, 2] \\ t_2 : [1, 5] \\ t_3 : [3, 3] \end{array}, 0, 0 \right)$

- $q_1 = \left(\begin{Bmatrix} p1 \\ p5 \\ p3 \end{Bmatrix}, t_1 : [0.6, 0.6], \mathbf{2}, 1.4 \times (2+3) = \mathbf{7} \right)$
- $q_2 = \left(\begin{Bmatrix} p4 \\ p5 \\ p3 \end{Bmatrix}, t_4 : [3, 3], 2+0 = \mathbf{2}, 7 + 0.6 \times (3+3) = \mathbf{10.6} \right)$
- $q_3 = \left(\begin{Bmatrix} p7 \\ p3 \end{Bmatrix}, t_6 : [1, 1], 2+3 = \mathbf{5}, 10.6 + 3 \times (3+3) = \mathbf{28.6} \right)$
- $q_4 = \left(\begin{Bmatrix} p9 \\ p3 \end{Bmatrix}, \mathit{none}, 5+1 = \mathbf{6}, 28.6 + 1 \times (5+3) = \mathbf{36.6} \right)$

At the end of the run we have $reward(\rho) = 6$ and $cost(\rho) = 36.6$

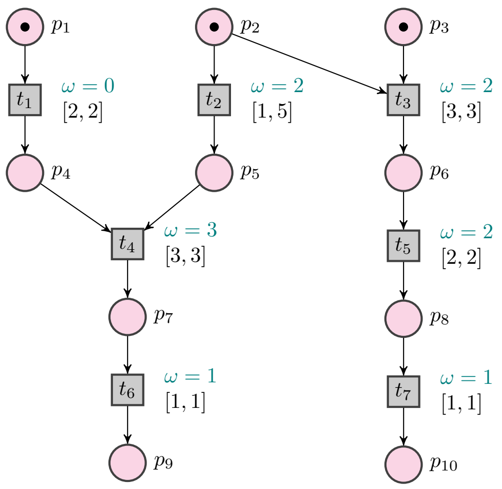


Figure 1 Cost Time Petri Net with $cr = 2 \times p_2 + 3 \times p_3 + 3 \times p_5 + 5 \times p_6 + 5 \times p_7 + 5 \times p_8 + 5 \times p_9$

5 Constrained-Cost problems and State Space

5.1 Constrained-Cost problems

Given a cTPN $\mathcal{N}$ and an upper-bound on the cost variable, the constrained-cost problems can be stated as following:

1. What is the optimal reward that can be reached without exceeding the upper cost limit?
2. Give a run that leads to this optimal reward and minimizes the cost

Definition 11 (Set of runs under cost constraint ($\mathsf{Runs}_{c \le c_{max}}$)) The set of runs of a cTPN $\mathcal{N}$ under the cost constraint $c \le c_{max}$ is the set of (finite or infinite) paths in its semantics starting in q_0 such that all the states of the run respect the constraint: $\rho = q_0 \xrightarrow{t_0 @ \delta_0} q_1 \xrightarrow{t_1 @ \delta_1} q_2 \cdots \xrightarrow{t_{n-1} @ \delta_{n-1} 2} q_n$ such that $\forall k \le n,\ \sum_{i=0}^{k-1} \delta_i * cr(m_i) \le c_{max}$

We denote this set by $\mathsf{Runs}_{c \le c_{max}}$

Definition 12 (Optimal reward of a cTPN under cost constraint) The optimal reward under the cost constraint $c \leq c_{max}$ is

$OptReward(c \leq c_{max}) = reward(\rho)$ such that $\rho \in \mathsf{Runs}_{c \leq c_{max}}$ and $\nexists \rho' \in \mathsf{Runs}_{c \leq c_{max}} \mid reward(\rho') > reward(\rho)$.

Definition 13 (Optimal run under cost constraint) An optimal run of the cTPN $\mathcal{N}$ under the cost constraint $c \leq c_{max}$ is

$OptRun(c \leq c_{max}) = \rho \in \mathsf{Runs}_{c \leq c_{max}}$ such that $reward(\rho) = OptReward(c \leq c_{max})$ and $\nexists \rho' \in \mathsf{Runs}_{c \leq c_{max}} \mid reward(\rho') = reward(\rho)$ and $cost(\rho') < cost(\rho)$.

5.2 State space under cost constraint

We now extend the state class of [26, 33] with information on cost and reward. We call *cost state classes* these extended state classes.

Given a sequence σ of transitions leading to a classic state class $C_\sigma = (m, D)$, the firing domain D is a convex polyhedron constraining the firing times of the transitions enabled by m. For an enabled transition t_i, we denote by θ_i the corresponding variable in D. These firing times are relative to the absolute firing date of the last transition of σ (or 0 for the initial class).

Cost state classes $L_\sigma = (m, \mathcal{R}, F)$ extend the state class as follows :

- the discrete state is now given by the marking m and the reward $\mathcal{R}$ obtained by the sequence of transitions σ,
- the firing domain is extended with an additional cost variable c, initially equal to c_0, and evolving as described in the semantics above, and using the following observation: since firing dates are relative to the last fired transition, the time spent in a class before firing some transition t_i is exactly θ_i.

Computing the successive cost state classes then naturally extends the classic computation of [26, 33] as follows:

- the initial cost state class is: $L_\varepsilon = (m_0, 0, \{\theta_i \in I_s(t_i) \mid t_i \in En(m_0)\} \wedge \{c = c_0\})$.
- A transition t_f is firable from class $L_\sigma = (m, \mathcal{R}, F)$ under cost constraint $c \leq c_{max}$ iff:
  - t_f is enabled by m;
  - $(F \wedge \bigwedge_{i \neq f} \theta_f \leq \theta_i \wedge c + \theta_f \times cr(m) \leq c_{max}) \neq \emptyset$.

  $\mathcal{F}irable(L_\sigma, c \leq c_{max})$ denotes the set of transitions t_f firable from L_σ under the cost constraint $c \leq c_{max}$
- The successor $L_{\sigma t_f}$ of the cost state class L_σ by a transition t_f firable from L_σ is given by Algorithm 1.

By iteratively computing the extended state classes we obtain a possibly infinite graph with edges labeled by fired transitions and nodes by classes.

Given a constraint $c \leq c_{max}$ where c_{max} is a finite integer, Algorithm 2 consists in a classic exploration of the symbolic state-space, while checking

Algorithm 1 Successor $L_{\sigma t_f} = (m', \mathcal{R}', F')$ of $L_\sigma = (m, \mathcal{R}, F)$ by firing t_f: $L_{\sigma t_f} = Next(L_\sigma, t_f)$

```
1: $m' \leftarrow m - {}^\bullet t_f + t_f^\bullet$;
2: $\mathcal{R}' \leftarrow \mathcal{R} + \omega(t_f)$;
3: $F' \leftarrow F \wedge \bigwedge_{i \neq f} \theta_f \leq \theta_i \wedge c + \theta_f \times cr(m)$;
4: for all $i \neq f$, add variable $\theta'_i$ to $F'$, constrained by $\theta_i = \theta'_i + \theta_f$;
5: add variable $c'$ to $F'$, constrained by $c' = c + \theta_f \times cr(m)$ ;
6: eliminate (by projection) variables $c$, $\theta_i$ for all $i$, and $\theta'_j$ for all $t_j$ disabled
   by firing $t_f$, from $F'$;
7: for all newly enabled transition $t_j$, add variable $\theta'_j$, constrained by $\theta'_j \in$
   $I_s(t_j)$.
```

the cost constraint in the firability condition. It uses a passed list, Passed, to store the already visited symbolic states.

The cost is not bounded in the Passed list, then the algorithm uses a dedicated comparison operator $\sqsubseteq$ between symbolic states [25, 34].

Definition 14 Let $L = (m, \mathcal{R}, F)$ and $L' = (m', \mathcal{R}', F')$ two cost state classes. We say that L is subsumed by L', which we denote by $L \sqsubseteq L'$ iff $m = m'$, $\mathcal{R} = \mathcal{R}'$ and $\uparrow F \subseteq \uparrow F'$ where $\uparrow F$ is the convex polyhedron obtained from F by removing all upper bound constraints on cost variable c.

We suppose that initially the cost is $c_0 \leq c_{max}$.

Algorithm 2 Symbolic algorithm for constrained-cost state space

```
1: ConstrainedStateSpace $\leftarrow \emptyset$
2: Passed $\leftarrow \emptyset$
3: Waiting $\leftarrow \{(m_0, 0, F_0 \wedge c = c_0)\}$
4: while Waiting $\neq \emptyset$ do
5:     select $L = (m, \mathcal{R}, F)$ from Waiting
6:     if for all $L' \in$ Passed, $L \not\sqsubseteq L'$ then
7:         add $L$ to Passed
8:         for all $t_f \in \mathcal{F}irable(L, c \leq c_{max})$, add $Next(L, t_f)$ to Waiting
9:     end if
10: end while
11: for each $L = (m, \mathcal{R}, F) \in$ Passed do
12:     add $(m, \mathcal{R}, F \wedge c \leq c_{max})$ to ConstrainedStateSpace
13: end for
14: return ConstrainedStateSpace
```

Firability condition $t_f \in \mathcal{F}irable(L, c \leq c_{max})$ checks $(F \wedge c + \theta_f \times cr(m) \leq c_{max}) \neq \emptyset$ then in all firing domain F' computed by the algorithm there is at least one point in F' such that $c \leq c_{max}$. However, we do not take into account the cost constraint in the computation of the successor $Next(L, t_f)$ therefore there may exist some domain F of reachable class $(m, \mathcal{R}, F)$ containing points in $F \wedge (c > c_{max})$. It is obvious that no transition is firable under cost constraint $c \leq c_{max}$ from such a point. Hence PASSED list computed by Algorithm 2 is correct w.r.t marking and reward but is an over-approximation of the firing domain. To obtain in CONSTRAINEDSTATESPACE the firing domain verifying the cost constraint from PASSED, it is therefore necessary to intersect the domains with $c \leq c_{max}$.

Algorithm 2 will not terminate if the number of reachable markings or rewards is not finite while the cost constraint is respected. We now prove the termination of the algorithm in case of boundedness.

Lemma 1 ($F_{\sigma|\theta} \equiv D_\sigma$) *Let $\mathcal{N}$ a cost TPN and $\mathcal{N}_U$ its underlying TPN (i.e., without cost and reward). Let σ a firable sequence in $\mathcal{N}$ from the initial state, leading to $L_\sigma = (m, a, F_\sigma)$ and $C_\sigma = (m, D_\sigma)$ respectively for $\mathcal{N}$ and $\mathcal{N}_U$. The projection of F_σ over the θ variables is $F_{\sigma|\theta} \equiv D_\sigma$.*

Proof By induction on the length n of the sequence σ: for n = 0 the property trivially holds. Suppose the lemma holds for σ with $L_\sigma = (m, a, F)$, $C_\sigma = (m, D)$ and $F_{|\theta} \equiv D$. Consider t a firable transition from C_σ leading to $C_{\sigma.t} = (m', D')$ then $F_{|\theta} \wedge \bigwedge_{i \neq f} \theta_f \leq \theta_i \neq \emptyset$ then $F \wedge \bigwedge_{i \neq f} \theta_f \leq \theta_i \neq \emptyset$ and t_f is firable from L_σ. Moreover, no constraint over c are added by the computation of $Next(L_\sigma, t_f)$ (Algorithm 1). The equation $c' = c + \theta_t \times cr(m)$ and the elimination of the variable c do not change the space of solutions over θ then ${F'}_{|\theta'} \equiv D'$. □

Therefore, as for the state class graph of definition 6, a firing domain $F_{\sigma|\theta}$ can be described by linear inequations of the form $\theta_i \leq k$ or $\theta_j - \theta_i \leq k'$ where $k \in \mathbb{N}$ and $k' \in \mathbb{Z}$.

Theorem 1 *In case the cost rates are integers, Algorithm 2 terminates if the underlying TPN is bounded and the reward is bounded.*

Proof If the underlying TPN is bounded then the number of reachable state classes $C = (m, D)$ of this underlying TPN is finite. Moreover the number of reward is also finite (bounded positive integer). Suppose that there is in the cost TPN reachable graph, an infinite number of cost state classes, then there is an infinite number of cost state classes sharing the same marking, the same reward and, from lemma 1, the same firing domain $F_{|\theta}$ of transitions. These cost state classes differ only on the constraints over the variable c. It has been proved in [25, 35] that the relaxed domain $\uparrow F$ of a state class of a cost TPN, can be partitioned into a union of simpler polyhedra with exactly one constraint on the cost variable $c \geq c_{min}$ with $c_{min} = \ell(\theta_1, \ldots, \theta_n)$ where ℓ is a linear function with integer coefficients. In case the cost rates are integers, these simple polyhedra have integer vertices of the form $((\theta_1, \ldots, \theta_n, \ell(\theta_1, \ldots, \theta_n))$ and since ℓ has integer coefficients, then c_{min} is an integer greater than zero. Moreover the firability condition guarantees that for all F computed by Algorithm 2, $F \cap (c \leq$

$c_{max}) \neq \emptyset$ then c_{min} is integer between 0 and c_{max} contradicting the assumption. □

5.3 Optimal reward and optimal runs from Algorithm 2

Recall that we aim to maximize the reward by respecting the maximal cost constraint but among the solutions, we will take the minimum cost leading to this maximal reward.

The optimal cost of the sequence σ leading to $L_\sigma = (m, a, F)$ is $inf(F_{|c})$ (*i.e.*, the minimal value of c in F). Since for all state classes in PASSED, we have $inf(F_{|c})$ $=inf((F \cap c \leq c_{max})_{|c})$, then CONSTRAINEDSTATESPACE and PASSED lists share the same minimal cost. Hence we can be satisfied with PASSED list to compute the solution of the problems defined in Section 5.1. Then we have:

$OptReward(c \leq c_{max}) = \mathcal{R}_{max} = max(\mathcal{R} \mid (m, \mathcal{R}, F) \in \text{PASSED})$

There may be several state classes that have the optimal reward and minimizing the cost. This set is: $OptL = \{L_\sigma = (m, \mathcal{R}, F_\sigma)$ such that $\mathcal{R} = \mathcal{R}_{max}$ and $inf(F_{\sigma|c}) = \min(inf\ (F_{|c}) \mid (m, \mathcal{R}_{max}, F) \in \text{PASSED})\}$

Finally, there may be several optimal runs:

$OptRun(c \leq c_{max}) = \{\rho$ such that $L_\sigma \in OptL$, $sequence(\rho) = \sigma$ and $cost(\rho)$=$inf(F_{\sigma|c})\}$

5.4 Example

Let us go back to the cTPN of Figure 1, page 13. The state class graph under the cost constrained $c \leq 30$ is given in Figure 2 (we omit in the figure the detail of classes). The initial state class is L_{σ_0}.

The firing of the sequences $\sigma_3 = t_1 t_2$ or $\sigma'_3 = t_2 t_1$ leads two different classes $L_{\sigma_3} = (m_3, \mathcal{R}_3, F_3)$ and $L_{\sigma'_3} = (m_3, \mathcal{R}_3, F'_3)$ sharing the same marking, reward and firing domain, with different cost but with the same minimum value of this cost. We give here the detail of classes L_{σ_0}, L_{σ_1}, L_{σ_2}, L_{σ_3} and $L_{\sigma'_3}$:

$$L_{\sigma_0} = \left(\left\{ \begin{matrix} p1 \\ p2 \\ p3 \end{matrix} \right\}, 0, \left\{ \begin{array}{l} \theta_1 \in [2,2] \\ \theta_2 \in [1,5] \\ \theta_3 \in [3,3] \\ -1 \leq \theta_2 - \theta_1 \leq 3 \\ -2 \leq \theta_3 - \theta_2 \leq 2 \\ 1 \leq \theta_3 - \theta_1 \leq 1 \\ c = 0 \end{array} \right. \right)$$

$$L_{\sigma_1} = \left(\left\{ \begin{matrix} p2 \\ p3 \\ p4 \end{matrix} \right\}, 0, \left\{ \begin{array}{l} \theta_1 \in [0,1] \\ c \in [5,10] \\ c \geq 10 - 5 * \theta_1 \end{array} \right. \right)$$

$$L_{\sigma_2} = \left(\left\{ \begin{matrix} p1 \\ p3 \\ p5 \end{matrix} \right\}, 2, \left\{ \begin{array}{l} \theta_2 \in [0,3] \\ \theta_3 \in [1,1] \\ -2 \leq \theta_3 - \theta_2 \leq 1 \\ c = 10 \end{array} \right. \right)$$

$$L_{\sigma_3} = \left(\left\{ \begin{matrix} p3 \\ p4 \\ p5 \end{matrix} \right\}, 2, \left\{ \begin{array}{l} \theta_4 \in [3,3] \\ c \in [10,15] \end{array} \right. \right) \quad L_{\sigma'_3} = \left(\left\{ \begin{matrix} p3 \\ p4 \\ p5 \end{matrix} \right\}, 2, \left\{ \begin{array}{l} \theta_4 \in [3,3] \\ c = 10 \end{array} \right. \right)$$

We have $F'_{3|c} \subseteq F_{3|c}$ and $\uparrow F_3 = \uparrow F'_3$ and then $L_{\sigma'_3} \sqsubseteq L_{\sigma_3}$ and $L_{\sigma_3} \sqsubseteq L_{\sigma'_3}$. These state classes will be merged by Algorithm 2 but depending on the order of exploration, the state class selected can be either L_{σ_3} or $L_{\sigma'_3}$.

The state class L_{σ_5}, in dashed on Figure 2, is not in the state graph because the c_{max} bound of 30 is exceeded.

We obtain $OptReward(c \leq c_{max}) = \mathcal{R}_{max} = 5$

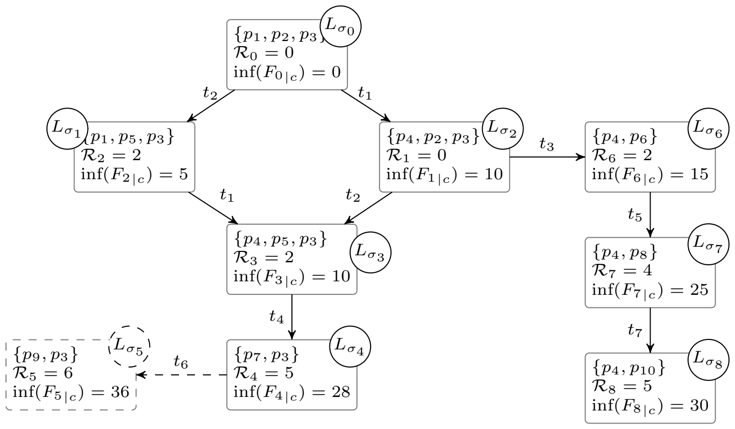
The classes L_{σ_4} and L_{σ_8} have this reward. Since $\mathcal{R}_4 = \mathcal{R}_8 = 5$ and $\inf(F_{4|c}) < \inf(F_{8|c})$, the optimal state class is L_{σ_4} where $\inf(F_{4|c}) = 28$.

Finally, from this state class graph and the goal state class, we can compute two similar solutions for $OptRun(c \leq 30)$:

- $\rho_1 = q_0 \xrightarrow{t_2@2} q_1 \xrightarrow{t_1@0} q_2 \xrightarrow{t_4@3} q_3 \xrightarrow{t_6@1} q_4$
- $\rho_2 = q_0 \xrightarrow{t_1@2} q'_1 \xrightarrow{t_2@0} q_2 \xrightarrow{t_4@3} q_3 \xrightarrow{t_6@1} q_4$

Note that in ρ_1, t_2 is fired at date 2 (then immediately before t_1) because the cost in p_2 is 2, which is lower than the cost in p_5 (which is 3). In ρ_2, t_2 is also fired at date 2 (then immediately after t_1) allowing to reduce the duration of the run and then the cost.

By imagining that for the system modeled, we can control the firing of t_2, the strategy to have an optimal run consists in firing t_2 at date 2.



L_{σ_0} $\{p_1, p_2, p_3\}$ $\mathcal{R}_0 = 0$ $\inf(F_{0|c}) = 0$
t_2
t_1
L_{σ_1} $\{p_1, p_5, p_3\}$ $\mathcal{R}_2 = 2$ $\inf(F_{2|c}) = 5$
L_{σ_2} $\{p_4, p_2, p_3\}$ $\mathcal{R}_1 = 0$ $\inf(F_{1|c}) = 10$
t_3
L_{σ_6} $\{p_4, p_6\}$ $\mathcal{R}_6 = 2$ $\inf(F_{6|c}) = 15$
t_1
t_2
t_5
L_{σ_3} $\{p_4, p_5, p_3\}$ $\mathcal{R}_3 = 2$ $\inf(F_{3|c}) = 10$
L_{σ_7} $\{p_4, p_8\}$ $\mathcal{R}_7 = 4$ $\inf(F_{7|c}) = 25$
t_4
t_7
L_{σ_5} $\{p_9, p_3\}$ $\mathcal{R}_5 = 6$ $\inf(F_{5|c}) = 36$
t_6
L_{σ_4} $\{p_7, p_3\}$ $\mathcal{R}_4 = 5$ $\inf(F_{4|c}) = 28$
L_{σ_8} $\{p_4, p_{10}\}$ $\mathcal{R}_8 = 5$ $\inf(F_{8|c}) = 30$

Figure 2 State Graph of cTPN of Fig. 1

6 On-the-fly algorithm to find a good candidate

We have presented an algorithm that computes the set of optimal runs of a cTPN based on an exhaustive exploration of its state space. This algorithm may suffer from state space explosion problems. Thus, we will now focus on a simpler problem: finding an optimal state class, *i.e.*, the final state class from an optimal run (*OptRun*) as defined in 5.3. To do so, we present first a greedy algorithm that computes a "good" solution with low computational complexity, and an exact algorithm.

Finding a cost-optimal run in the cost-constrained state space is akin to finding a shortest path in a directed graph. In fact, the algorithm that we propose takes an approach similar to that of A^*, a well-known algorithm [36] to solve the shortest-path problem in a directed graph without having to explore the whole graph. To do this, A^* uses a function to estimate the cost of the shortest path between the root of the graph and the destination that passes through a given node n. This function is usually noted f, with $f(n) = g(n) + h(n)$ where g is the function that gives the (known) cost between the source and n, and h is a heuristic that estimates the (unknown) cost between n and the destination. A^* explores the graph by ordering the node by increasing value of f and stops as soon as it reaches the destination. If h is pessimistic, *i.e.*, it yields an upper-bound on the actual cost, then A^* is optimal (*i.e.*, it is guaranteed to find the shortest path).

However, our problem is not exactly a shortest-path problem. First, in our case, the destination is not identified in advance. We stop the exploration as soon as we encounter a state class that has no successor in the space of constrained-cost state space. Second, we have not one but two variables in our optimization problem: the cost we want to minimize, and the reward we want to maximize. As in A^*, we propose to guide the exploration of the constrained-cost state space by using a function f which we use to order the state classes according to their relevance to our problem. However, in our case, this function does not give an estimate of the cost of a run (because we have two variables), and is not separated into two parts g and h. The two techniques we propose in the following paragraphs are both based on traversing the constraint-cost state space as described in the 3 algorithm. They differ by:

- f, the function used to direct the exploration,
- and NEEDTOEXPLORE, the procedure to choose which successors of a given state class need to be considered.

Heuristic search procedure based on maximal reward per cost ratio

In the first version of the analysis, the exploration order is driven by the cost per reward ratio. Let R_n the reward accumulated between the source and state class n, and c_n the minimum cost to reach n. Value $f(n)$ is computed as follows:

$$f(n) = \begin{cases} \frac{c_n}{\mathcal{R}_n} & \text{if } \mathcal{R}_n \neq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{3}$$

The constrained-cost state space will be examined by exploring the successors of the state class that has the best "dynamic" among the class present in the WAITING list. All successors should be examined but those that have already been met (see algorithm 4)

Algorithm 3 Heuristic-based algorithm for finding an optimal state class in the constrained-cost state space.

```
1: PASSED ← $\emptyset$
2: WAITING ← $\{(m_0, 0, F_0 \wedge c = c_0)\}$
3: while WAITING $\neq \emptyset$ do
4:     select $L = (m, \mathcal{A}, F)$ whose value of $f$ is the smallest from WAITING
5:     if $\mathcal{F}irable(L, c \leq c_{max}) = \emptyset$ then
6:         return $L$
7:     end if
8:     remove $L$ from WAITING
9:     add $L$ to PASSED
10:    for $t_f \in \mathcal{F}irable(L, c \leq c_{max})$ do
11:        $L' = Next(L, t_f)$
12:        if NEEDTOEXPLORE($L'$, PASSED) then
13:            add $L'$ to WAITING
14:        end if
15:    end for
16: end while
```

Algorithm 4 Function NEEDTOEXPLORE for the greedy algorithm.

```
procedure NEEDTOEXPLORE($L$, PASSED)
    return $L \notin$ PASSED
end procedure
```

Since the algorithm greedily follows the highest reward/cost ratio, it is not guaranteed to return a state class that is part of an optimal run. However, following the highest reward/cost ratio is a common-sense strategy to move quickly through the constrained-cost state space in order to obtain a good approximate solution, in a reasonable amount of time and computation.

According to the cTPN of Figure 1, using this first version of the analysis, the part of the constrained-cost state space which is explored is given Figure 3. In this example, the analysis finds the optimal solution, but it is not always the case. For instance, as $f(L_{signa_2}) = +\infty$, its successors will never be explored, even if the optimal solution was among them.

Heuristic search procedure based on a discretization of the cost

In the problem defined in section 3.3, the *reward* is strongly related to the cost. For any cost there is a maximum *reward* associated such that $\mathcal{R}_i \leq k \times c_i$, with k an integer. As it is true for all pairs of points $(\mathcal{R}, c)$, it is also true for $(\mathcal{R}_{max}, c_{max})$ and by knowing $\mathcal{R}_{max}$ we can deduce k. In such a model, on a graph *reward*/cost, the *reward* evolves according to a staircase function (each time a transition with a non-zero reward is fired), and will never exceed $k \times c$,

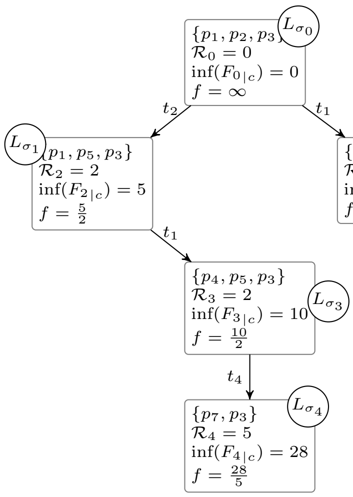


Figure 3 State Graph of cTPN of Fig. 1 using greedy algorithm

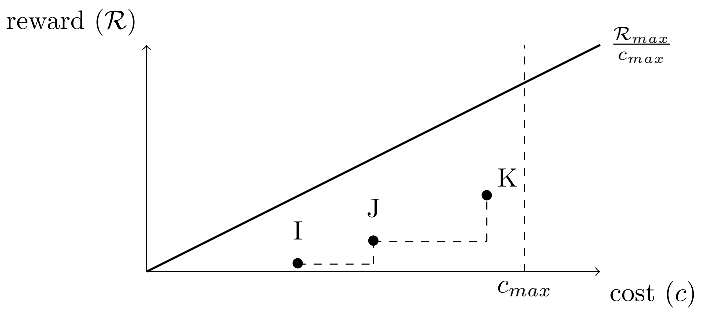


Figure 4 Example of a case where the reward is a discretization of the cost

where c is the cost accumulated until now. This is illustrated in Figure 4. In this example, J is a successor of I and K is a successor of J. The slope of the line passing through I and J, as well as the slope of the line passing through J and K is less than or equal to the slope $\frac{\mathcal{R}_{max}}{c_{max}}$.

The second analysis is designed for systems where the reward is a discretization of the cost. For these systems, we define $\mathcal{R}_{max}$ as an upper-bound of the reward for all runs. This value is such that in every state, the overall reward per cost ratio is less than or equal to $\frac{\mathcal{R}_{max}}{c_{max}}$.

This new piece of information can now be used to guide the exploration of the constrained-cost state space. To do so, we compute $f(n)$ as follows:

$$f(n) = \begin{cases} \frac{1}{\mathcal{R}_n + \frac{\mathcal{R}_{max}}{c_{max}} \times (c_{max} - c_n)} & \text{if } c_n \neq c_{max} \text{ and } \mathcal{R}_n \neq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{4}$$

With this function, the search procedure is now driven by the actual reward of the state class and the best hypothetical reward that can be obtained by a run passing through it. Thus, when exploring the constrained-cost state space, we can safely ignore the successors of state class Lj whenever the following criterion holds:

$$\exists L_n \in \textsc{Passed}, L_n = (m_n, \mathcal{R}_n, F_n), \quad \frac{1}{\mathcal{R}_n} < \frac{1}{\mathcal{R}_j + \frac{\mathcal{R}_{max}}{c_{max}} \times (c_{max} - c_j)} \tag{5}$$

Whenever the criterion holds, it means that the best possible reward that could be obtained by exploring successors of L_j will be smaller than the reward of a class already explored. Indeed, from criterion (5), one can trivially prove that : $\mathcal{R}_n > \mathcal{R}_j + \frac{\mathcal{R}_{max}}{c_{max}} \times (c_{max} - c_j)$.

On a reward/cost graph, this can be highlighted by drawing a straight horizontal line from the intersection between the vertical line c_{max} and the projection of j by the parallel line of slope $\frac{\mathcal{R}_{max}}{c_{max}}$ starting from j. On Figure 5, any node that has a reward/cost point on the triangle formed by $\triangle ABC$ (for example node i) is already a better solution than any successor of node j.

The criterion (5) is added in algorithm 3 in the needToExplore function as described in algorithm 5. Unlike the heuristic defined for the greedy algorithm, we ensure that the optimal state is achieved as it explores all states except those removed by the criterion. In the worst-case scenario, the procedure explores all classes of the constrained-cost state graph.

Algorithm 5 Function needToExplore for the exact algorithm.

```
1:  procedure needToExplore($L$, Passed)
2:      if $L \in$ Passed then
3:          return False
4:      end if
5:      for M in Passed do
6:          if $\frac{1}{\mathcal{R}_M} < f(L)$ then
7:              return False
8:          end if
9:      end for
10:     return True
11: end procedure
```

Using the example from 1, the exploration will be as the state graph in Figure 6. In this case, the heuristic will not cut any branch as in any state, the criterion is never satisfied. The heuristic would cut states after L_{σ_6} in the case where the heuristic value of f of L_{σ_6} would be less than $\frac{1}{5}$ (from the state L_{σ_4} where $\frac{1}{\mathcal{R}_n} = \frac{1}{5}$)

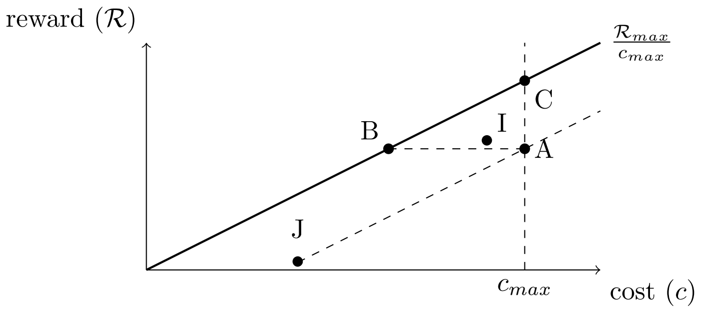


Figure 5 Graph elimination of successors of a node with Criterion (5)

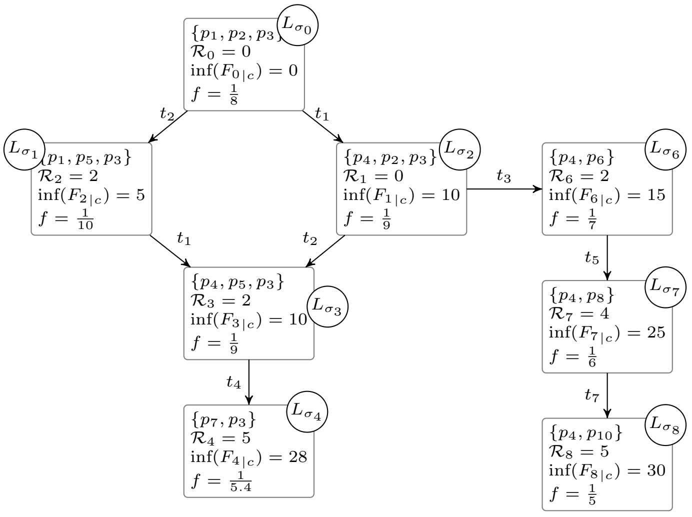


Figure 6 State Graph of cTPN of Fig. 1 using heuristic from equation (4)

7 Case Study - Reactive Intermittent System

For our case study, we have chosen a bird song recognition application. This type of application is fairly representative of intermittent edge computing because the IoT node is deployed in an area that requires autonomous power and maintenance can become very expensive, such as changing the battery. It can also be tolerated that the system does not work permanently because the purpose here is to measure bird population densities to monitor migrations without establishing an accurate count. Finally, a large amount of computation is required to recognize a song, and the computation should be performed on the edge to reduce the size of the information transmitted over the wireless network. Ideally, this system consists of an energy harvesting mechanism, coupled with a small energy buffer such as a super-capacitor to harvest and store energy. The part performing the calculation is composed of a micro-controller and different peripherals for data acquisition, processing and sending via a wireless network.

The prototype is built around a Texas Instruments evaluation board, the Launchpad MSP-EXP430FR5994. A board of our design is plugged on top and includes 1MB additional FRAM, accessible via the SPI bus, a microphone with its amplifier and a LoRa radio transceiver. The processing pipeline includes signal sampling, song recognition using an FFT and finally, sending the result of the recognition by radio if successful.

7.1 Energy consumption model

As explained in the section 3.1, in the absence of a voltage regulator, the evolution of the super-capacitor voltage is linear. A measurement campaign was carried out on the prototype board and the evolution of the voltage is given in 3 different cases on the figure 7. To model the prototype, we first identified a set of modes. For each mode identified, we then designed a benchmark, which allowed us to measure the voltage across the super-capacitor when the system is in that mode. Each run started from an initial state where the super-capacitor is fully charged and stopped when the voltage dropped below 1.9V, the low operating limit of the micro-controller. To control these experiments and to carry out the measurements in a non-intrusive way, we have created an *ad-hoc* circuit board. In all, we have characterized 38 modes, but other modes can easily be added, including modes linked to external devices. For each mode, we have made 5 voltage measurements over time. The results presented are averaged to obtain a voltage slope for the given mode.

We can notice that the 3 measurements give a linear evolution of the voltage as a function of time, as expected. The slope varies from -8.9 mV/s when the processor, the ADC and the DMA module are active, to -2.9 mV/s when only the processor is active.

7.2 Modeling an intermittent system

Mode.

We use a notion of operating mode of an intermittent system similar to what is introduced in the previous work [2]. A mode is a state of the system characterized by the list of active sub-systems, as well as by their voltage slope. The same sub-system can have different voltage slopes. For example, a peripheral can have more than one voltage slope according to different input. Thus, for each mode, we associate a voltage slope of the system which is obtained by adding the voltage slopes of the active sub-systems, and which informs us about the dynamic power of the system in this mode. We can then model the execution of an intermittent system as a state machine, where each transition marks the activation or deactivation of one or more sub-systems. The sub-systems considered are the micro-controller components, as well as the internal and external peripherals.

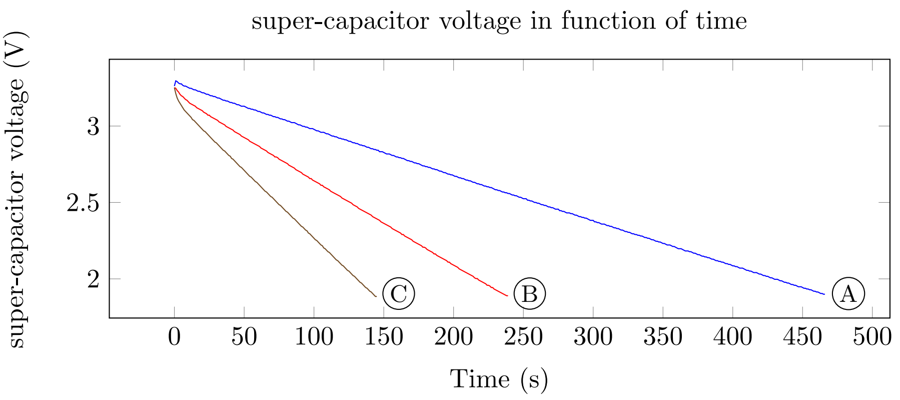


Figure 7 Empirical measure of voltage slope for sub-systems of an intermittent platform. The capacity of the super-capacitor is 0.22F. From right to left: (A) only the CPU is active, (B) the CPU and ADC are active, (C) the CPU, ADC and DMA are active. The system stops working when the voltage drops below 1.9V.

We use the cost variable from the cTPN to model the energy consumption of the system. Each mode in the system has a place associated with it. A token is placed in this slot when the mode becomes active, and removed when it becomes inactive. These token movements are controlled by the part of the model that describes program execution and device behavior. When a token is positioned in a mode place, the dynamics of the cost variable is changed: it increases with a slope corresponding to the voltage slope of the mode. It is of course possible for several modes to be active simultaneously, the global dynamics being then obtained by addition.

Consider the example from figure 8 which models an application composed of two tasks. When the transition $ChooseTask_1$ is fired, the system starts to execute task 1 and the mode $Mode_1$ becomes active. Similarly, when the $ChooseTask_2$ transition is fired, the system starts executing task 2 and the mode $Mode_2$ becomes active. The rate of change of the cost variable cr is then defined by : $cr = \sum_m slopeMode_m \times Mode_m$, where $slopeMode_1 = 5$ and $slopeMode_2 = 2$ in the example.

The cost variable therefore increases, at each unit of time, by the value of the voltage slope of the various active modes. In this example, the modes are used to model the energy consumption induced by the execution of a task as a function of its execution time.

The reward ω for each task is obtained at the end of the execution, thus associated to the transition $TerminateTask_m$.

Energy/execution duality.

Firstly, one might want to use the cost to model the energy of our intermittent system and minimize its consumption. However, for an intermittent system, minimizing energy consumption is not necessarily the answer. On the one hand, if energy is not used, it is lost and we would rather use the available energy

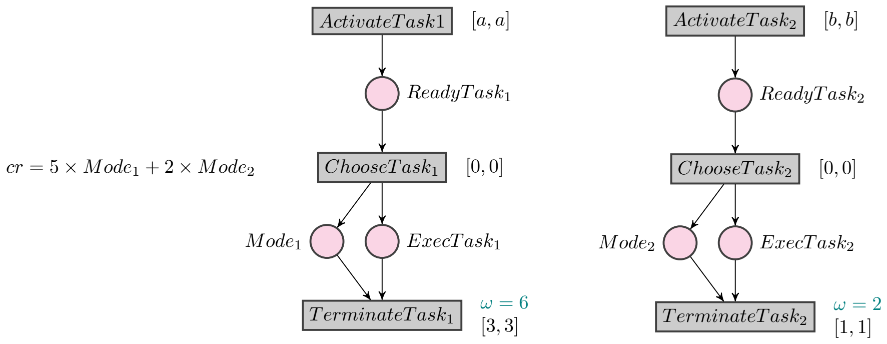


Figure 8 Modelling of 2 simple tasks associated with a mode, using a timed cost Petri net. The cost increases when a token is in places $Mode_1$ and/or $Mode_2$, with a different slope for each.

as much as possible. On the other hand, minimizing the energy used can lead to overly aggressive pruning in the model, with energy-intensive tasks never being called, even though they are essential for the proper functioning of the whole system.

7.3 Trace synthesis

We have used the modeling elements presented previously to model a complete case study.

We have built a model of the system in Roméo, a toolbox for modeling and verifying TPNs. Roméo already supports the notion of cost, but not that of reward. So we used Romeo to compute the state space of the underlying cost TPN. Then, we computed the set of states within this state space allowing to reach the best reward using ad hoc scripts. Finally, we again used Romeo to compute the minimum cost path to reach these maximum reward states. This is a first naive and sub-optimal implementation of our approach, in order to establish its feasibility. The integration of our algorithms within Roméo will be the subject of future work.

In the initial state, the super-capacitor is full. Each task is associated with a mode as described in section 7.2. For readability, *modes* places and actions related to them are not shown. The Petri net model of the application can be found in annex A. The execution is as follows:

1. Periodic data acquisition: sound samples using the microphone.
2. Data processing: use of a hardware accelerator for FFT[4] (Fast Fourier Transform) operations and according to the energy available :
    - store the result in an internal non-volatile buffer (transition *BufferFFT*), then sleep or
    - continue to analysis stage(transition *GoToClassif*)..

[4]The MSP430FR5994 micro-controller includes a hardware accelerator dedicated to signal processing operations.

3. Data analysis : Sound analysis algorithm based on FFT results.
4. Finalization according to available energy :
    - store the result in an external non-volatile buffer, then sleep (transition *FramExt*) or
    - transmit the result over a wireless link (transition *Send*).

External non-volatile memory has a larger capacity than internal non-volatile memory but it costs more energy to use. 2 memory buffers to store FFT results are allocated in the internal volatile memory. 10 memory buffers to store recognition results are allocated in the external volatile memory (for readability, only 5 tokens are displayed). Each processed data occupies a single token in the buffer. The transmission of data cannot be interrupted. The goal of the application is to send processed data, so we apply a high reward to sending data and a lower reward to storing processed or unprocessed data.

In the Petri net model, we force the application to stop only in predefined states to mimic the checkpoint mechanism. For example, doing a checkpoint after the acquisition of audio data from a microphone requires storing all the data recorded in NVM (otherwise, data will be deleted), thus it would be more interesting to perform the checkpoint when the data is processed as it will be more memory-friendly (*i.e.*, after the FFT or the sound analysis algorithm). The state where the application is able to checkpoint safely are places with name starting with *checkpoint* on the Petri net.

Some transition are controllable with interval between 0 and ∞. These are the transitions that send data from buffer to the next stage of the application (*i.e.*, *FFTtoClassif* and *SendDataFromFramExt* transitions). Other intervals are fixed and they reflect the worst-case execution time of the task (WCET).

The model described is roll-back free as any task interrupted is reset when the power come back. As the energy harvested is not determined or used in the model, we compute traces to optimal states using several upper-bounds of the cost to mimic different energy level available from a state. On the target, this would correspond to the energy harvested while the system is running or by a task that ran faster than its WCET (different inputs lead to different execution times). In both cases, the energy available when the system has reached the optimal state according to the synthesized trace can be different.

The synthesized traces are given in the form of a graph where the nodes are the system states and the transitions are the traces leading to the different states. An example is provided in appendix B. For readability, traces are not displayed, but for example, trace t_1 corresponds to the transitions : PeriodAudio Acquisition FFT GoToClassif Classification GenerateFrame Send PeriodAudio Acquisition SendLoRa.

Another level of analysis from these results is to prevent unfinished tasks from starting in order to save energy. For example with trace t_1, the second acquisition of data (*i.e.*, transitions *PeriodAudio Acquisition* after transition *Send*) will never reach the next checkpoint and data corresponding to this execution is lost, meaning energy is wasted. The microphone can be switched off after the first acquisition to avoid unnecessary power consumption.

The graph is computed with 2 energy levels (variable E , *i.e.*, 15 and 10). These energy levels can be more finely discretized, but this will generate more traces and these traces will be longer.

The goal of these traces is to provide information on the system to help designers. For example, on the model of the application from appendix A, traces obtained can help to define an optimal size for the buffers used.

Using those traces online can also provide static scheduling according to both energy available and actual state of the system.

8 Conclusion

This paper presented an optimal trace synthesis using Cost Time Petri Nets (cTPN) to manage the energy buffer of intermittent systems. The consumption of the system is modeled, depending on the operating mode, by a linear cost versus time. The progress of the application is characterized by rewards associated with functional goals such as producing a result or sending it by radio. The whole is combined in a cTPN specifying the tasks and functional dependencies. Two heuristics allowing to avoid exploring the whole state space are proposed and allow to synthesize traces that can then be exploited on an execution platform and a case study of a prototype bird song recognition system presents an example of traces synthesis.

Future work will focus on the implementation of the synthesized optimal traces. They will be used to implement an ad-hoc scheduler in an existing RTOS which will be modified accordingly. This execution support will be deployed on the prototype with the application.

Declarations

The authors declare that they have no conflict of interest.

Acknowledgement : This work has received a French government support granted to the Labex CominLabs excellence laboratory and managed by the National Research Agency in the ”Investing for the Future” program under reference ANR-10-LABX-07-01

References

[1] Ransford, B., Sorber, J. & Fu, K. *Mementos: System support for long-running computation on rfid-scale devices*, ASPLOS XVI, 159–170 (Association for Computing Machinery, 2011).

[2] Balsamo, D. *et al.* Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embed. Syst. Letters* **7** (1) (2015). https://doi.org/10.1109/LES.2014.2371494 .

[3] Balsamo, D. *et al.* Hibernus++: A self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* **35** (12) (2016). https://doi.org/10.1109/TCAD.2016.2547919 .

[4] Maeng, K., Colin, A. & Lucia, B. Alpaca: Intermittent execution without checkpoints. *Proc. ACM Program. Lang.* **1** (2017). https://doi.org/10.1145/3133920 .

[5] Colin, A. & Lucia, B. *Termination checking and task decomposition for task-based intermittent programs* (2018).

[6] Yarahmadi, B. & Rohou, E. *Compiler Optimizations for Safe Insertion of Checkpoints in Intermittently Powered Systems* (2020). URL https://hal.inria.fr/hal-02914953.

[7] Berthou, G., Delizy, T., Marquet, K., Risset, T. & Salagnac, G. Sytare: A lightweight kernel for nvram-based transiently-powered systems. *IEEE Trans. Comput.* **68** (9) (2019) .

[8] Ruppel, E. & Lucia, B. *Transactional concurrency control for intermittent, energy-harvesting computing systems* (2019).

[9] Yıldırım, K. S. *et al.* *InK: Reactive kernel for tiny batteryless sensors* (2018).

[10] Maeng, K. & Lucia, B. *Adaptive low-overhead scheduling for periodic and reactive intermittent execution*, PLDI 2020, 1005–1021 (Association for Computing Machinery, 2020).

[11] Berthou, G., Dagand, P., Demange, D., Oudin, R. & Risset, T. *Intermittent computing with peripherals, formally verified* (2020).

[12] Surbatovich, M., Lucia, B. & Jia, L. Towards a formal foundation of intermittent computing. *Proc. ACM Program. Lang.* **4** (OOPSLA) (2020). URL https://doi.org/10.1145/3428231. https://doi.org/10.1145/3428231 .

[13] Wägemann, P., Dietrich, C., Distler, T., Ulbrich, P. & Schröder-Preikschat, W. *Whole-System Worst-Case Energy-Consumption Analysis for Energy-Constrained Real-Time Systems*, Vol. 106 (2018).

[14] Sliper, S. T., Wang, W., Nikoleris, N., Weddell, A. S. & Merrett, G. V. *Fused: Closed-loop performance and energy simulation of embedded systems*, 263–272 (2020).

[15] Dezfouli, B., Amirtharaj, I. & Li, C. EMPIOT: an energy measurement platform for wireless iot devices. *CoRR* **abs/1804.04794** (2018). URL http://arxiv.org/abs/1804.04794. arXiv:1804.04794 .

[16] Berthou, G., Marquet, K., Risset, T. & Salagnac, G. *Accurate power consumption evaluation for peripherals in ultra low-power embedded systems* (2020).

[17] Kansal, A., Hsu, J., Zahedi, S. & Srivastava, M. B. Power management in energy harvesting sensor networks. *ACM Trans. Embed. Comput. Syst.* **6** (4) (2007) .

[18] Cammarano, A., Petrioli, C. & Spenza, D. Online energy harvesting prediction in environmentally powered wireless sensor networks. *IEEE Sensors Journal* **16** (17) (2016) .

[19] Alur, R., Torre, S. L. & Pappas, G. J. Optimal paths in weighted timed automata. *Theoretical Computer Science* **318** (3), 297 – 322 (2004). URL http://www.sciencedirect.com/science/article/pii/S0304397503005838. https://doi.org/http://dx.doi.org/10.1016/j.tcs.2003.10.038 .

[20] Abdulla, P. A. & Mayr, R. Priced timed petri nets. *Logical Methods in Computer Science* **9** (4) (2013). URL http://dx.doi.org/10.2168/LMCS-9(4:10)2013. https://doi.org/10.2168/LMCS-9(4:10)2013 .

[21] Behrmann, G. *et al.* *Minimum-Cost Reachability for Priced Timed Automata*, 147–161 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001). URL http://dx.doi.org/10.1007/3-540-45351-2_15.

[22] Larsen, K. *et al.* As cheap as possible: Efficient cost-optimal reachability for priced timed automata. *Lecture Notes in Computer Science* **2102**, 493–505 (2001) .

[23] Behrmann, G., Larsen, K. G. & Rasmussen, J. I. Optimal scheduling using priced timed automata. *SIGMETRICS Perform. Eval. Rev.* **32** (4), 34–40 (2005). URL https://doi.org/10.1145/1059816.1059823. https://doi.org/10.1145/1059816.1059823 .

[24] Lime, D., Roux, O. H., Seidner, C. & Traonouez, L.-M. Kowalewski, S. & Philippou, A. (eds) *Romeo: A parametric model-checker for Petri nets with stopwatches.* (eds Kowalewski, S. & Philippou, A.) *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, Vol. 5505 of *Lecture Notes in Computer Science*, 54–57 (Springer, York, United Kingdom, 2009).

[25] Boucheneb, H., Lime, D., Parquier, B., Roux, O. H. & Seidner, C. Nestmann, U. & Wolter, K. (eds) *Optimal reachability in cost time Petri nets.* (eds Nestmann, U. & Wolter, K.) *15th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2017)*, Vol. 10419 of *Lecture Notes in Computer Science*, 58–73 (Springer, Berlin, Germany, 2017).

[26] Berthomieu, B. & Diaz, M. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Software Eng.* **17** (3), 259–273 (1991) .

[27] Bouyer, P., Larsen, K. G. & Markey, N. Lower-bound-constrained runs in weighted timed automata. *Performance Evaluation* **73**, 91–109 (2014). Special Issue on the 9th International Conference on Quantitative Evaluation of Systems .

[28] Boukhobza, J., Rubini, S., Chen, R. & Shao, Z. Emerging NVM: A survey on architectural integration and research challenges. *ACM Trans. Design Autom. Electr. Syst.* **23** (2), 14:1–14:32 (2018). https://doi.org/10.1145/3131848 .

[29] Haritsa, J. R., Carey, M. J. & Livny, M. Value-based scheduling in real-time database systems. *VLDB J.* **2** (2), 117–152 (1993). URL http://www.vldb.org/journal/VLDBJ2/P117.pdf .

[30] Jain, N., Menache, I., Naor, J. & Yaniv, J. A truthful mechanism for value-based scheduling in cloud computing. *Theory Comput. Syst.* **54** (3), 388–406 (2014). URL https://doi.org/10.1007/s00224-013-9449-0. https://doi.org/10.1007/s00224-013-9449-0 .

[31] Prasad, D. & Burns, A. A value-based scheduling approach for real-time autonomous vehicle control. *Robotica* **18** (3), 273–279 (2000). https://doi.org/10.1017/S0263574799002349 .

[32] Merlin, P. M. *A study of the recoverability of computing systems.* Ph.D. thesis, Department of Information and Computer Science, University of California, Irvine, CA (1974).

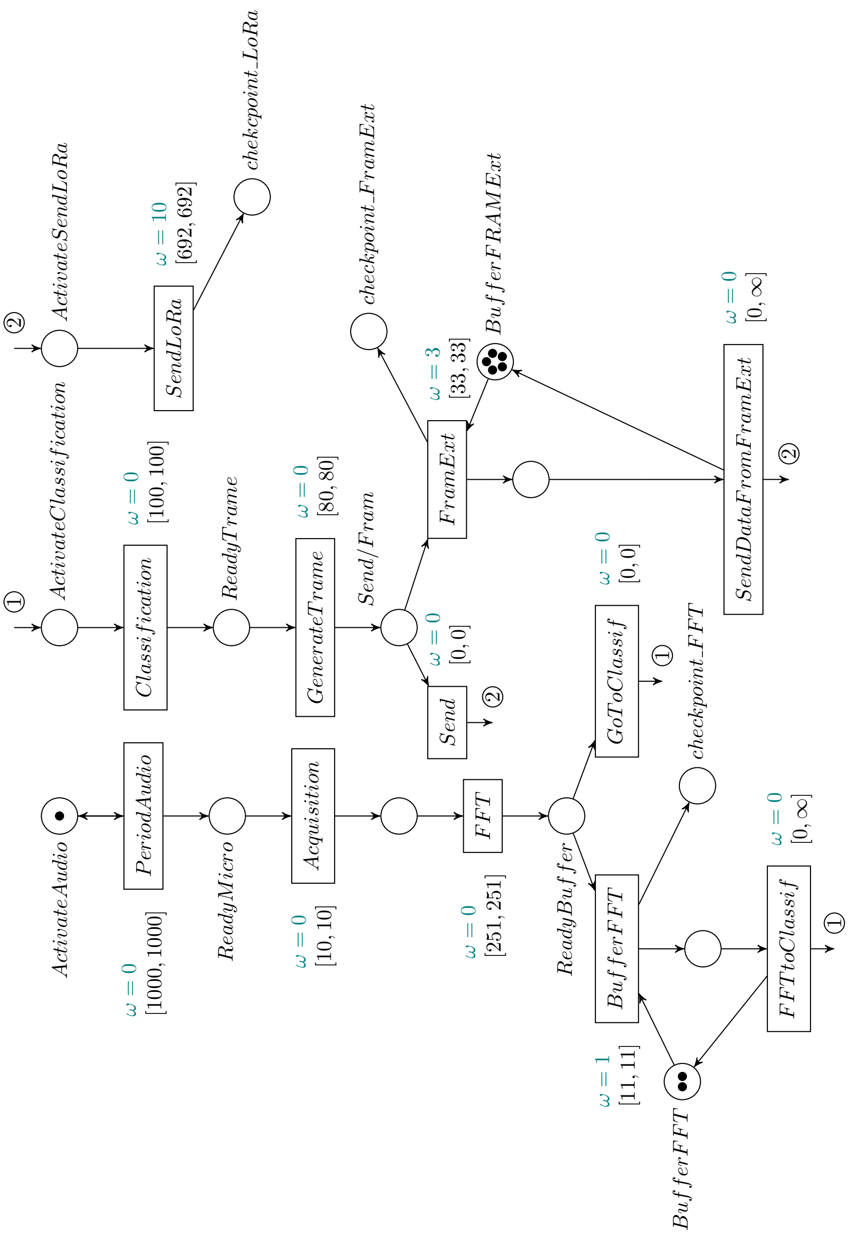
[33] Berthomieu, B. & Menasche, M. *An enumerative approach for analyzing time petri nets*, 41–46 (1983).

[34] Rasmussen, J. I., Larsen, K. G. & Subramani, K. On using priced timed automata to achieve optimal scheduling. *Formal Methods in System Design* **29** (1), 97–114 (2006) .

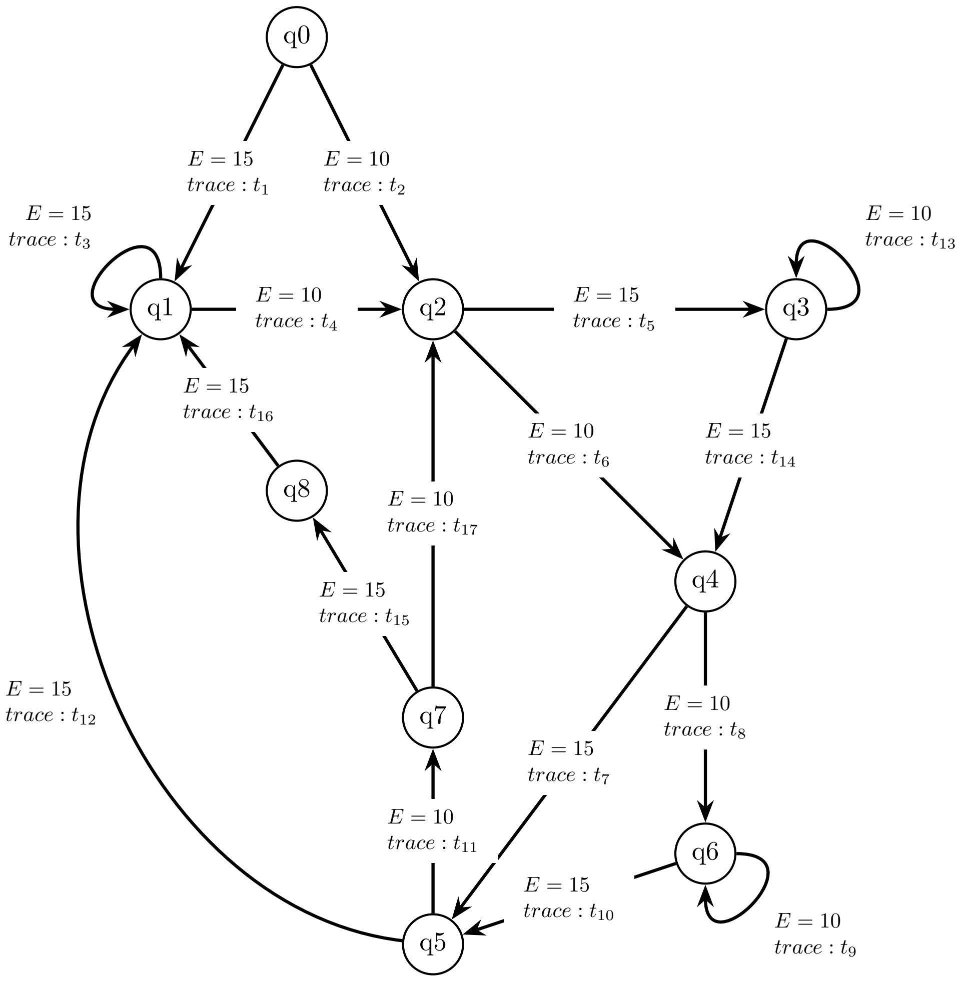
[35] Lime, D., Roux, O. H. & Seidner, C. Cost problems for parametric time petri nets. *Fundamenta Informaticae* **183** (1-2), 97–123 (2021) .

[36] Hart, P. E., Nilsson, N. J. & Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* **4** (2), 100–107 (1968). https://doi.org/10.1109/TSSC.1968.300136, conference Name: IEEE Transactions on Systems Science and Cybernetics .

A Petri net model of case-study



B Synthesized traces for case-study


q0
q1
q2
q3
q4
q5
q6
q7
q8
$E = 15$ $trace : t_1$
$E = 10$ $trace : t_2$
$E = 15$ $trace : t_3$
$E = 10$ $trace : t_4$
$E = 15$ $trace : t_5$
$E = 10$ $trace : t_6$
$E = 15$ $trace : t_7$
$E = 10$ $trace : t_8$
$E = 10$ $trace : t_9$
$E = 15$ $trace : t_{10}$
$E = 10$ $trace : t_{11}$
$E = 15$ $trace : t_{12}$
$E = 10$ $trace : t_{13}$
$E = 15$ $trace : t_{14}$
$E = 15$ $trace : t_{15}$
$E = 15$ $trace : t_{16}$
$E = 10$ $trace : t_{17}$