Incremental Integer Linear Programming Models for Petri Nets Reachability Problems

Thomas Bourdeaud'huy[1], Saïd Hanafi[2] and Pascal Yim[1]
[1]*L.A.G.I.S. Ecole Centrale de Lille*
[2]*L.A.M.I.H. Université de Valenciennes*
France

1. Introduction

The operational management of complex systems is characterized, in general, by the existence of a huge number of solutions. Decision-making processes must be implemented in order to find the best results. These processes need suitable modeling tools offering true practical resolution perspectives. Among them, Petri nets (PNs) provide a simple graphical model taking into account, in the same formalism, concurrency, parallelism and synchronization. Their graphical and precise nature, their firm mathematical foundation and the aboundance of analysis methods have made them become a classical modeling tool for the study of discrete event systems, ranging from operating systems to logistic ones. However, their interest in the field of problem solving is still badly known.

In this paper, we consider some PN *reachability problems*. Since PNs can model flows in a natural and efficient way, many operations research problems can be defined using reachability between states, e.g. scheduling (Lee and DiCesare, 1994; Van Der AAlst, 1995), planning (Silva et al., 2000), car-sequencing problems (Briand, 1999). Moreover, research on Petri nets addresses the issue of flexibility: many extensions have been proposed to facilitate the modeling of complex systems, by addition of ``color'', ``time'' and ``hierarchy'' (Jensen, 1992; Wang,1998). For example, it is relatively easy to map scheduling problems onto timed PNs. Their graphical nature reinforce obviously this strength, by allowing a kind of *interactivity* with the system. At last, a large number of difficult PN analysis problems are equivalent to the reachability problem, or to some of its variants or sub-problems (Keller, 1976). Particularly, *model-checking* (Latvala, 2001) which represents a key point when dealing with systems analysis is *directly* linked to an exhaustive traversal of the corresponding PN reachability graph.

Various methods have been suggested to handle the PNs reachability problem. In this paper, we propose to use the mathematical programming paradigm. Some PN analysis problems have already been handled using such techniques (Melzer and Esparza, 1996;Silva et al., 1998; Khomenko and Koutny, 2000), but none has considered the general PNs reachability problem.

The proposed approach is based on an *implicit traversal* of the Petri net reachability graph, which does not need its construction. This is done by considering a unique sequence of *steps* growing incrementally to represent exactly the total behavior of the net. We follow here a

Source: Petri Net,Theory and Applications, Book edited by: Vedran Kordic, ISBN 978-3-902613-12-7, pp. 534, February 2008, I-Tech Education and Publishing, Vienna, Austria

previous work from (Benasser and Yim, 1999) called *logical abstraction technique*. Their technique was validated on several examples using logical constraint programming techniques. It has shown more effective than other generic solvers and could even compete with heuristics dedicated to particular classes of problems. Our methodology allows to improve this original model using the wide range of tools and adjustments brought by *Operational Research* techniques. We model the problem as an integer linear program, then we solve it with a branch-and-bound technique (divide and conquer), using the Cplex optimization software.
Moreover, we show how our incremental approach can be extended to *Timed Petri nets* in order to solve scheduling problems modelled as Timed Petri Nets reachability problems. The model built is as general as possible since we do not make assumptions about the firing policy, contrarywise to other classical approaches dealing with the same issue.
This chapter is organized as follows. In section 2, we formally define the kind of PN considered, their respective reachabilty problems and the ways such problems are dealt with in the litterature. Then, in section 3, we give general considerations about step firings and describe the elements of our incremental approaches. In section 4, we apply our methodology to express reachability problems using a mathematical programming formulation. Finally, as a conclusion, we describe a few promising research directions.

2. Petri Nets reachability problems

In this section, we give the terminology of both kinds of the PN we are interested in using linear algebra -- in order to make our formulations more concise -- and define formally their respective reachability problems.

2.1 Place/transition Petri nets
2.1.1 Petri net terminology

Definition 1 (Place/Transition Petri Net). *A Place/Transition Petri net (Murata, 1989)* $R = (\mathrm{P}, \mathrm{T}, C^-, C^+)$ *with its initial marking* m *is a bipartite weighted directed graph where:*

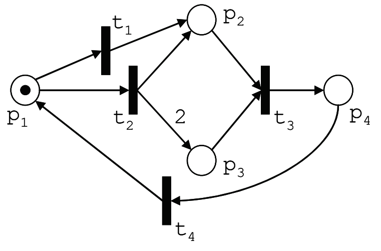
- $\mathrm{P} = \{p_1, \ldots, p_m\}$ *is a finite set of places, with* $M = |\mathrm{P}|$. *Places are represented as circles and indexed by letter* i*;*
- $\mathrm{T} = \{t_1, \ldots, t_n\}$ *is a finite set of transitions, with* $N = |\mathrm{T}|$. *Transitions are represented as rectangles and indexed by letter* j*;*
- *Incidence matrices* C^-, C^+ *and* $C \in \mathrm{N}^{\mathrm{P}\times\mathrm{T}}$ *(with* $C = C^+ - C^-$*) define the weighted flow function which associates to each arc* (p_i, t_j) *(from place* p_i *to transition* t_j*) or* (t_j, p_i) *(from transition* t_j *to place* p_i*) its weight* C_{ij}^- *or* C_{ij}^+. *When there is no arc between place* p_i *and transition* t_j*, then we have:* $C_{ij}^- = C_{ij}^+ = 0$. *The* i^{th} *row vector and* j^{th} *column vector taken from incidence matrices* C^-*,* C^+ *and* C *are denoted respectively* C_i^-*,* C_j^-*,* C_i^+*,* C_j^+ *and* C_i*,* C_j*. We denote respectively by* ${}^\bullet p$ *and* $p^\bullet$ *the set of predecessors and*

successors of place p, and conversely ${}^{\bullet}t$ and $t^{\bullet}$ are the set of predecessors and successors of transition t (also known as input and output nodes);

- $m : \mathrm{P} \rightarrow \mathrm{N}$ *associates to each place $p \in \mathrm{P}$ an integer $m(p)$ called the marking of the place p. Markings are represented as full dots called tokens inside places.*

Definition 2 (Characteristic Vectors). *Let (R, m) be a Petri net with $\mathrm{P} = \{p_1, p_2, \ldots, p_m\}$ and $\mathrm{T} = \{t_1, t_2, \ldots, t_n\}$:*

- *The canonical vector $\overrightarrow{e_{p_i}}$ associated to place p_i (resp. $\overrightarrow{e_{t_j}}$ associated to transition t_j) is the vector in $\{0,1\}^N$ (resp. in $\{0,1\}^M$) which takes the value ``1'' in its $i^{th}$ (resp. $j^{th}$) component and ``0'' elsewhere.*
- *The marking vector $\overrightarrow{m}$ associated to marking m is the column vector $(m(p_1), m(p_2), \ldots, m(p_m))^{\text{ú}} \in \mathrm{N}^M$.*



$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 0 & 1 \\ 1 & 1 & -1 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

$$C^{-} \qquad\qquad C^{+} \qquad\qquad C$$

Fig. 1. A Petri Net and its Incidence Matrices

Example 1 (PN). *An example of a PN and its incidence matrices is presented in Fig.1. Its initial marking is $m_0 = (1,0,0,0)^{\text{ú}}$. We have $\overrightarrow{e_{p_2}} = (0,1,0,0,0)^{\text{ú}}$ and $\overrightarrow{e_{t_3}} = (0,0,1,0,0)^{\text{ú}}$.*

In a PN, the markings of the places represent the state of the corresponding system at a given moment. This state can be modified by the *firing* of transitions. This behaviour is called the ``*token game''*.

Definition 3 (Transition Firings). *Let (R, m) be a Petri net. A transition t_j is fireable from marking m iff:*

$$\begin{array}{rrcl} & \forall p_i \in \mathrm{P}, \quad m(p_i) & \geq & C^{-}_{ij} \\ \Leftrightarrow & \overrightarrow{m} & \geq & C^{-} \cdot \overrightarrow{e_{t_j}} \end{array} \tag{1}$$

The fireability condition is denoted by $m[t\rangle$. If this condition is satisfied, a new marking m' is produced from the marking m, such that:

$$\begin{array}{rrcl} & \forall p_i \in \mathrm{P}, \quad m'(p_i) & = & m(p_i) - C^-_{ij} + C^+_{ij} \\ \Leftrightarrow & \overrightarrow{m'} & = & \overrightarrow{m} + C \cdot \overrightarrow{e_{t_j}} \end{array} \tag{2}$$

The firing of a transition t from the marking m to the marking m' is denoted by $m[t\rangle m'$.
Transition firings modify the marking of the net. It is thus interesting to know if one particular marking can be reached. This problem is known as the ``*reachability problem*'' for Petri nets.

2.1.2 Reachability problem

Definition 4 (Reachable Marking). *A marking m' is reachable from a marking m iff there exists a sequence of transitions $\sigma = t_{\sigma_1} t_{\sigma_2} \dots t_{\sigma_k}$ such that: $m[t_{\sigma_1}\rangle m_1[t_{\sigma_2}\rangle m_2 \dots [t_{\sigma_K}\rangle m'$*

We denote by $m[\sigma\rangle m'$ that the marking m' is reachable from the marking m, where $\sigma = t_{\sigma_1} t_{\sigma_2} \dots t_{\sigma_K}$ is called a *firing sequence*. The *Parikh vector* $\overrightarrow{\sigma} = \sum_{k=1}^{K} \overrightarrow{e_{t_{\sigma_k}}}$ associated to the firing sequence σ is the vector whose j^{th} component is equal to the number of times the transition j is fired in σ. It is used to formulate a well known property of Petri Nets.

Proposition 1 (State equation). *Let (R, m_0) be a Petri net, m_f a marking and $\sigma = t_{\sigma_1} t_{\sigma_2} \dots t_{\sigma_K}$ a firing sequence. Then we have:*

$$m_0[\sigma\rangle m_f \Rightarrow \overrightarrow{m_f} = \overrightarrow{m_0} + C \cdot \overrightarrow{\sigma} \tag{3}$$

Proof. It is obtained using a simple induction over the number of transitions fired in the sequence. W

The equation **Error! Reference source not found.** is called the *fundamental (or state) equation* of Petri nets. This equation has been widely studied in PN reachability analysis, but it only leads to semi-decision algorithms due to the existence of *spurious solutions* (Silva et al., 1992). Indeed, in that case, the reverse implication does not hold: the Parikh vector of a firing sequence is always solution to the state equation, but the reverse is not true. Some techniques (Colom and Silva, 1989b) have been proposed to improve the strength of this characterization, but they are still insufficient.

Definition 5 (Reachability Problem). *Let (R, m_0) be a Petri net and m_f a marking. The set of all markings reachable from m_0 is denoted by $\mathrm{R}\,(R, m_0)$; the set of all possible firing sequences (within which each transition is fireable from the corresponding marking) is denoted by $\mathrm{F}\,(R, m_0)$. The problem of finding whether $m_f \in \mathrm{R}\,(R, m_0)$ or not is known as the reachability problem for Petri nets.*

It has been shown that the reachability problem is decidable (Kosaraju, 1982). However it is EXP-TIME and EXP-SPACE hard in the general case (Lipton, 1976). Of course, practical applications need not only to know if a marking is reachable, but also what are the corresponding firing sequences leading to this marking. To solve this problem, one needs to find a firing sequence $\sigma \in \mathrm{F}\ (R, m_0)$ such that $m_0[\sigma\rangle m_f$. A ``naive'' approach consists in exploring the *reachability graph* exhaustively. This graph corresponds to the usual formal representation of the behavior of the net.

Definition 6 (Reachability Graph). *The reachability graph of a Petri net* (R, m_0)*, denoted by* $\mathrm{G}(R, m_0)$*, is defined by:*

A set of nodes $\mathrm{R}\ (R, m_0)$ which represents the reachable markings;

A set of arcs, where an arc (m, m') labelled t connects nodes m and m' iff $m[t\rangle m'$.

Example 2 (Reachability Graph). *Fig.2. presents a part of the reachability graph for the Petri net of Fig.1.*

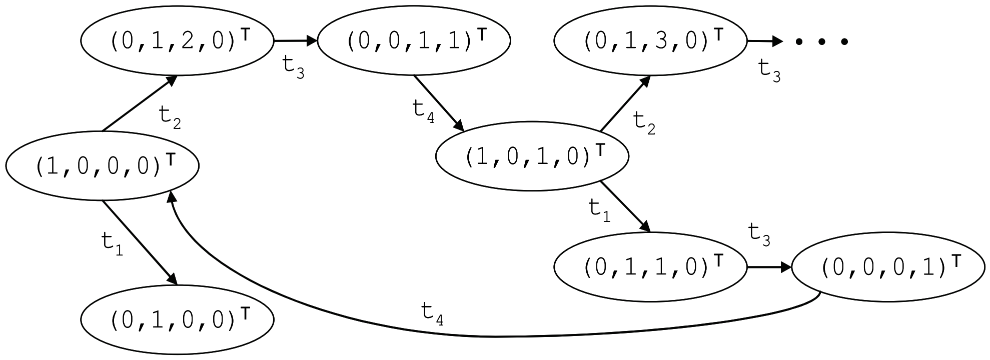


Fig. 2. Reachability graph for the PN of Fig. 1

For a given initial marking m_0, the reachability graph $\mathrm{G}(R, m_0)$ and the corresponding reachability set $\mathrm{R}\ (R, m_0)$ may be of infinite size. For instance, the set of markings reachable from m_0 for the net of Fig. 1 is infinite.

Practically, it is not possible to explore the reachability graph exhaustively due to the well known problem of *combinatorial explosion*: the size of the state-space (i.e. the size of the reachability set) may grow exponentially with the size of a system configuration (i.e. the number of nodes of the Petri net). Many methods have been studied to limit this explosion. Let us mention the three main families.

First ones aims at *managing* the combinatorial explosion without modifying the studied reachability graph. Classical approaches are *graph compressions*, particularly *bdd encoding* (Gunnarsson, 1998) and *forward checking* (Fernandez et al., 1992). Both uses depth first traversal of the reachability graph.

- Other techniques construct a *reduced* reachability graph associated to the original, based on some properties to preserve: symmetries (Huber et al., 1985), reductions (Berthelot, 1986) and partial order (*covering step graphs* (Vernadat et al., 1996), *stubborn sets*

(Valmari, 1991)) are the main approaches. The *logical abstraction technique* (Benasser and Yim, 1999) belongs also to this category.

- Last ones are based on the PN state equation (cf. Proposition 1): we can distinguish *parametrized analysis* (Lindqvist, 1993) and *algebraic methods* (Lautenbach, 1987).

Many extensions have been proposed to improve the modelling power of Petri nets. Among them, several extended Petri nets with ``time'' have been proposed by assigning punctual firing times (leading to ``*Timed PN''*) or time intervals (``*Time PN''*) to the components of Petri nets (transitions, places, arcs or even tokens). To deal with firing times, two main methods for modeling timing are used: either the timings are associated with the places (the PN is said to be P-timed) (Sifakis, 1975), or the timings are associated with the transitions (the PN is said to be T-timed) (Ramchandani, 1974). Depending on the system to be modeled, one of the models (P-timed or T-timed) may be easier to use than the other one. However, Sifakis has shown that the two models are equivalent. In the context of scheduling problems, (Hillion and Proth, 1989) and (Van Der Aalst, 1995) propose to use T-timed Petri nets, hereafter called simply *Timed PN*. We describe this model in the following section.

2.2 Timed Petri nets

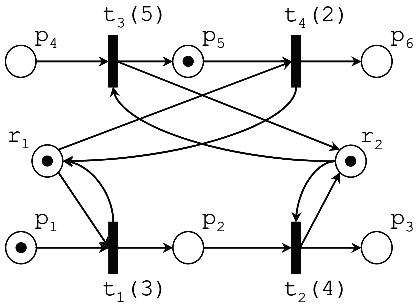
Timed Petri nets have been introduced by (Ramchandani, 1974). The following presentation has been adapted from (Chrétienne, 1984). We start by giving an informal introduction on Timed Petri nets.

2.2.1 Informal presentation

Timed Petri nets correspond to Places/Transitions Petri nets where a *duration* $d(t) \in \mathrm{N}^*$ is associated to each transition t. A Timed Petri net has the same representation as PN, to which is added a *labelling* on transitions. An example of Timed Petri net is given in Fig. 3. We have: $d(t_1) = 3$, $d(t_2) = 4$, $d(t_3) = 5$, $d(t_4) = 2$.

The firing durations associated to transitions modify the *marking validity conditions*. As soon as durations are associated to transitions, the Petri net acts as if tokens ``*disappeared''* at the time the transition is fired, and then ``*reappeared''* after a delay corresponding to the duration of the fired transition. Thus, the marking of a Timed Petri net evolves with the occurences of an external timer. For instance, let's consider the Timed Petri net of Fig. 3. At date 1, the transition t_1 (duration: *3 t.u.*) is fired. Then the transition t_4 (duration: *2 t.u.*) is fired at date 5. The evolution of marking with time is given in Fig. 3. Note that one could have fired transition t_4 at date 4, since the resource r_1 had been released at the end of the firing of transition t_1. However, the same transition was not fireable at date 3, since the firing of t_1 was not finished.

The firing and ending dates of transitions play a fundamental role in the behaviour of the Timed Petri net. It is thus necessary to *adapt* the firing equations according to these firing dates. In order to respect the underlying semantic of PN, a *timed firing sequence* is said to be *feasible* if and only if, at any time, the transient marking reached is made of *non negative* components.



	Date	Marking $(p_1,p_2,p_3,p_4,p_5,p_6,r_1,r_2)^{ú}$
Initial date	0	$(1,0,0,0,1,0,1,1)^{ú}$
Firing of $t_1 \rightarrow$	1	$(0,0,0,0,1,0,0,1)^{ú}$
	2	$(0,0,0,0,1,0,0,1)^{ú}$
	3	$(0,0,0,0,1,0,0,1)^{ú}$
End of $t_1 \rightarrow$	4	$(0,1,0,0,1,0,1,1)^{ú}$
Firing of $t_4 \rightarrow$	5	$(0,1,0,0,0,0,0,1)^{ú}$
	6	$(0,1,0,0,0,0,0,1)^{ú}$
End of $t_4 \rightarrow$	7	$(0,1,0,0,0,1,1,1)^{ú}$
	8	$(0,1,0,0,0,1,1,1)^{ú}$

Fig. 3. Example of a Timed Petri Net and a Timed Firing Sequence

2.2.2 Timed Petri nets terminology

Definition 7 (TPN -- Timed Petri Net). *A Timed Petri net (Ramchandani, 1974) is defined by a pair* (R,d) *where* R *is a Place/Transition Petri Net and* $d:\mathrm{T}\rightarrow \mathrm{N}^*$ *is a mapping associating a duration to each transition of the net. The vector* $\overrightarrow{d} = \sum_{t\in \mathrm{T}} d(t)\cdot \overrightarrow{e_t}$ *is called the duration vector of the Timed Petri net.*

Note that one could more generally consider *rational valued* durations. Nevertheless, after having them reduced to the same denominator, and by reasoning over numerators, it is the same as if durations were *integer valued*. In addition, to simplify the study, we restrict ourselves to Timed Petri nets *without immediate transitions* (i.e. $\forall t \in \mathrm{T}, d(t) > 0$), which is not so restrictive in real world practice and corresponds well to scheduling problems we are concerned with.

The transition firing semantics in TPN forbids *reentrance*. In other words, it is not possible to fire again a transition that has not yet *finished* to be fired. Again, this semantics is well fitted to scheduling problems, where transitions are associated to operations on machines. Thus, one can associate a unique *residual duration* to each transition without any possible confusion between several concurrent transitions activations. The residual duration vector is associated to the marking of a TPN to define its full state.

Definition 8 (TPN State). *Let* (R,d) *be a TPN. Its state* $e = (\overrightarrow{E_m}, \overrightarrow{E_r})$ *is given by:*

- Its classical marking vector $\overrightarrow{E_m} \in \mathrm{N}^M$, associating to each place its number of tokens;
- A residual durations vector $\overrightarrow{E_r} \in \mathrm{N}^N$, associating to each *active* transition its remaining duration, and zero if the transition is not active.

The set of all states of a TPN is denoted by $\mathrm{S}(R,d)$. The fundamental concept that governs Timed Petri net behavior is the *controlled execution*, which associates to each transition the sequence of its successive firing dates.

Definition 9 (CE – Controlled Execution) *Let (R,d) be a TPN and $t \in \mathrm{T}$ a transition. A firing sequence for the timed transition t: $(u_k^t) = u_1^t,\ldots,u_{K_t}^t \in \mathrm{N}$ is an increasing sequence of firing dates, such that:*

$$\forall k \in [\![1, K_t - 1]\!], u_k^t + d(t) \le u_{k+1}^t \qquad (4)$$

A controlled execution is a family $(u_k^t)_{t\in\mathrm{T},k\in[\![1,K_t]\!]}$ of firing sequences for all transitions of the TPN.

Note that in the previous definition, equation (4) is used to forbid *reentrance*. For any transition t, k_t and $u_{K_t}^t$ may be infinite. Hereafter, we only consider finite CEs. We denote by $v_{\max}$ the ending date of the last firing in the CE: $v_{\max} = \max\limits_{t\in\mathrm{T}}\left(u_{K_t}^t + d(t)\right)$. After $v_{\max}$, the state of the TPN under the considered CE will never change and we have: $\overrightarrow{E_r(v_{\max})} = \overrightarrow{0_N}$.

The formal expression of a CE is used to define several *characteristic vectors* allowing to verify the feasability of a CE. We assume that no transition is active at the initial state to simplify the formulation.

Definition 10 (Characteristic Vectors of Controlled Executions) *Let (R, d) be a TPN with its initial state $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ given at initial date 0 and $(u_k^t)_{t\in\mathrm{T},k\in[\![1,K_t]\!]}$ a controlled execution. Let $v \in [\![0, v_{\max}]\!]$. We define three characteristic vectors associated to (u_k^t) in the following way:*

$\overrightarrow{N(v)} \in \mathrm{N}^N$ is the vector corresponding to the number of firings that started within the interval $[0,v]$, defined by $\left.\overrightarrow{N(v)}\right|_t = card\left(\left\{u_{k,\,k\in[\![1,K_t]\!]}^t \mid u_k^t \le v\right\}\right)$;

- $\overrightarrow{D(v)} \in \mathrm{N}^N$ is the vector corresponding to the number of firings that started within the interval $[0,v[$, defined by $\left.\overrightarrow{D(v)}\right|_t = card\left(\left\{u_{k,\,k\in[\![1,K_t]\!]}^t \mid u_k^t < v\right\}\right)$;

- $\overrightarrow{F(v)} \in \mathbb{N}^N$ is the vector corresponding to the number of firings that ended within the interval $[0, v]$, defined by $\overrightarrow{F(v)}\Big|_t = card\left(\left\{u^t_{k,k\in[\![1,K_t]\!]} \mid u^t_k + d(t) \le v\right\}\right)$.

We have introduced above the definitions of *state* and *controlled execution* of a TPN. We define below how the state of a TPN is modified under a CE.

Definition 11 (Instantaneous State of a TPN under a Controlled Execution) *Let (R, d) be a TPN with its initial state $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ given at date 0 and $(u^t_k)_{t\in T, k\in[\![1,K_t]\!]}$ a controlled execution. Let $v \in [\![0, v_{\max}]\!]$. The instantaneous state $e_v = \left(\overrightarrow{E_m(v)}, \overrightarrow{E_r(v)}\right)$ at date v is given by:*

$$\overrightarrow{E_m(v)} = \overrightarrow{E_{m_0}} + C^+ \cdot \overrightarrow{F(v)} - C^- \cdot \overrightarrow{N(v)} \tag{5}$$

$$\forall t \in T, \overrightarrow{E_r(v)}\Big|_t = \begin{cases} u^t_k + d(t) - v & \text{if } \exists k \in [\![1, K_t]\!]\ s.t.\ v \in [\![u^t_k, u^t_k + d(t)[\![\\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Informally, in the previous definition, the quantity $C^+ \cdot \overrightarrow{F(v)}$ corresponds to the tokens produced by the firings of transitions that ended strictly before the date v. Those tokens can be used to fire transitions *at date* v. The quantity $C^- \cdot \overrightarrow{N(v)}$ corresponds to the tokens used by the firings of transitions that started *until the date* v. Thus, the quantity $\overrightarrow{E_m(v)}$ corresponds exactly to the tokens remaining in the TPN at date v. The residual durations vector $\overrightarrow{E_r(v)}$ denotes the exact remaining time of transitions that are active at date v. Obviously, there can only be one $k \in [\![1, K_t]\!]\ s.t.\ v \in [\![u^t_k, u^t_k + d(t)[\![$ from equation (4). Note that (Chretienne, 1984) defines also the quantity $\overrightarrow{m^-(v)} = \overrightarrow{E_m(0)} + C^+ \cdot \overrightarrow{F(v)} - C^- \cdot \overrightarrow{D(v)}$. This quantity does not consider the tokens used by the firings of transitions that occur *exactly at date* v. Thus, it can be used to formulate the fireability condition for a transition in a TPN, independently from possible concurrent activations: under a controlled execution, a transition is fireable at date v iff $\overrightarrow{m^-(v)} \ge C^- \cdot \overrightarrow{e_t}$.

Obviously, like for Place/Transitions PNs, even if each transition is independently fireable at every date, the full CE is not necessarily valid as a whole since token may be used by several transitions at the same time. Thus, an improved condition for a CE to be feasible is given below.

Definition 12 (Feasible Controlled Execution). *Let* (R,d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* 0 *and* $(u_k^t)_{t\in \mathrm{T},\, k\in[\![1,K_t]\!]}$ *a controlled execution. This controlled execution is said to be feasible iff:*

$$\forall v \in [\![0, v_{\max}]\!], \overrightarrow{E_m(v)} \geq \overrightarrow{0_M} \tag{7}$$

The previous condition means that there must be enough tokens so that transitions may fire simultaneously.

2.2.3 Timed Petri Net Reachability Problem

Using the previous notations, the Timed Petri nets reachability problem consists in searching for a feasible CE allowing to reach a given final state from the initial state.

Definition 13 (Timed PN Reachability Problem). *Let* (R,d) *be a TPN with its initial state* $e_0 = \left(\overrightarrow{E_{m_0}}, \overrightarrow{0_N}\right)$ *given at date* 0. *Let* $e_f = \left(\overrightarrow{E_{m_f}}, \overrightarrow{0_N}\right)$ *be a target state. The reachability problem for Timed Petri nets consists in finding a CE* $(u_k^t)_{t\in \mathrm{T},\, k\in[\![1,K_t]\!]}$ *such that* $e_{v_{\max}} = \left(\overrightarrow{E_m(v_{\max})}, \overrightarrow{E_r(v_{\max})}\right) = e_f$.

As said before, it is quite simple to see a parallelism between a scheduling problem and a Timed Petri net reachability problem. Indeed, let's consider for instance the Timed Petri net of Fig. 4. One remarks obviously that solving a reachability problem between markings $m_0 = \{p_1, p_2, p_3, m_1, m_2, m_3\}$ and $m_f = \{p_4, p_5, p_6, m_1, m_2, m_3\}$ means exactly finding a schedule of the production presented in the table on the left side.

Production To Schedule:

job 1:
$(m_1,2)(m_2,3)(m_3,4)$

job 2:
$(m_2,3)(m_3,2)(m_1,3)(m_3,3)$

job 3:
$(m_1,2)(m_3,4)(m_1,2)$

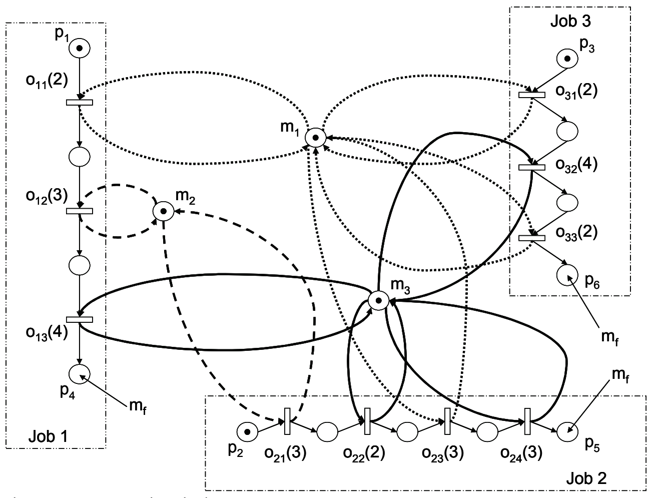


Fig. 4. TPN modelling a Production To Schedule

Several approaches have been proposed to solve the Timed Petri net reachability problem, either by restricting their study to a subclass of TPN, like Timed Event Graphs (where a place has exactly one input and one output transition), either by using dedicated heuristics. A complete bibliography can be found in (Richard, 2000).
Since the fire of a Timed transition can occur as soon as it is fireable and as late as one wants, there may exist, from a given state, an infinite number of reachable markings (depending on the time), and no reachability graph can be built. A first approach needs to consider Timed PN as a subclass of Time PN, in order to use the state enumeration methods (*state class graphs*) proposed by (Berthomieu and Diaz, 1991). On the other hand, when dealing with *early semantics* (a transition is fired as soon as it is fireable), it is possible to proceed to an enumerative and structural analysis (David and Alla, 1992).
The early semantics has been extensively studied for the special class of Timed Event Graphs, using (max,+) algebra (Bacceli et al., 1992). Since their structure does not handle conflicts, it is possible to obtain linear equations corresponding to the complete behaviour of the net.
In the following, we will show that our incremental approach can lead to mathematical programming models in the *most general* case.

3. Incremental approaches

As said before, the state equation (3) does not bring enough information to solve the reachability problem in all cases. This comes from the fact that it does not take into account the fireability conditions (1) of the individual transitions fired in the sequence σ. Incremental approaches improve this formulation by considering a given number of *step firings* corresponding to *parallel* and *reentrant* transitions. In this section, we discuss the interest of using steps and a fixed depth formulation.

3.1 Step based reachability formulation

Definition 14 (Step). *Let* R *be a Petri net. A step (Janicky and Koutny, 1991) is a multiset over the set of transitions* T *. We denote by* T^* *the set of steps built over* T *.*
Informally, a step is a set that can contain several copies of the same element, e.g. $\{t_1, t_1, t_2\}$, which we would note hereafter simply $2 \cdot t_1 + t_2$. We associate a step $\varphi = \sum_{j=1}^{N} \alpha_j \cdot t_j$ and its Parikh vector $\overrightarrow{\varphi}$ in the classical way, as a linear combination with non negative integer coefficients α_j of the Parikh vectors of each transition, i.e. $\overrightarrow{\varphi} = \sum_{j=1}^{N} \alpha_j \cdot \overrightarrow{e_{t_j}}$. A step is said *empty*, when $\varphi = \varnothing$, i.e. when $\forall j \in \llbracket 1, N \rrbracket, \alpha_j = 0$.
Note that a step can contain the same transition more than once, corresponding to *transition reentrance*. Thus, when working with Timed Petri nets, steps would only mean that several *different* transitions are considered to be fired at the same time.
For a step to be fireable, its preceding marking must contain enough tokens so that each transition of the step may consume its *own* tokens, as described in the following definition.

Definition 15 (Step Firings). *Let R be a Petri net, m be a marking and φ be a step . The step φ is fireable from m iff:*

$$\overrightarrow{m} \geq C^{-}.\overrightarrow{\varphi} \tag{8}$$

If this condition is satisfied, the new marking m' reached from m by the firing of φ is defined as:

$$\overrightarrow{m'} = \overrightarrow{m} + C\,.\overrightarrow{\varphi} \tag{9}$$

Hereafter, we will use the notations already used previously: $m[\varphi\rangle$, $m_0[\varphi\rangle m_1$, $m_0[\varphi_1\varphi_2\ldots\varphi_k\rangle$ and $m_0[\varphi_1\varphi_2\ldots\varphi_k\rangle m_k$ to denote that a step or a step sequence is fireable, and the marking obtained in each case. The number of steps of a step sequence $\Phi = \varphi_1\varphi_2\ldots\varphi_K$ is denoted by $|\Phi| = K$.

The definition of step firings corresponds naturally to the firing of the underlying transitions. We will show that its use can lead to a formulation that is still equivalent to the initial PN behavior, but that can be more conveniently used in a mathematical programming framework. The following proposition explains the relation between step and transition firings with respect to reachability issues.

Proposition 2 (Step Reachability Equivalence). *Let (R, m_0) be a Petri net and m_f a marking.*

m_f is reachable from $m_0 \Leftrightarrow$

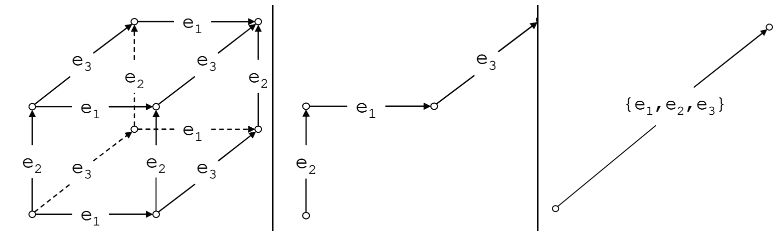
$$\begin{array}{l}\exists k \in \mathrm{N}, \\ \exists m_1, m_2, \ldots, m_{k-1} \in \mathrm{N}^M, \\ \exists \varphi_1, \varphi_2, \ldots, \varphi_K \in \mathrm{T}^*\end{array} \; s.t.: \begin{cases} \forall k \in [\![1, K-1]\!], & m_{k-1}[\varphi_k\rangle m_k \\ \wedge & m_{k-1}[\varphi_K\rangle m_f \end{cases} \tag{10}$$

Proof. The proof of this proposition is not difficult but quite lengthly and hence is not given in this chapter. It can be found in a technical report available at url: http://www.eclille.fr/tomnab/asr07/. W

One must remark that the proof of the proposition 2 shows how it is possible to *construct* a firing sequence leading to m_f from a corresponding step sequence. Thus, to compute a firing sequence leading to a target marking, it will be sufficient to compute a step sequence leading to the same marking.

The main interest of our formulation is to *capture the parallelism* caused by the *interleaving of actions*, which is precisely one of the main advantages of using a Petri net as a model of a system. This issue has already been followed by (Vernadat et al., 1996), from whom we borrow the illustrative example of Fig. 5: ``*When two (n in the general case) components offer independent actions in parallel, the interleaving semantics expresses this behaviour by a diamond (an hypercube in the general case) within which each path lead to the same final state. Since all paths* *converge to the same state, the key idea is to develop only one particular path among the set of possible equivalent ones".*

Example 3 (Vernadat's steps). *As shown in Fig. 5, there exists several ways to handle 3 independent transitions from the point of view of the reachability graph. One can consider them one by one, which leads to handle 8 states and 12 firings. If we just refer to their corresponding Mazurkiewicz's trace (see (Vernadat et al., 1996) for details), we only have to handle 3 transition firings and 4 states. In the last case, one can capture the whole behavior in one unique firing that is called a step by Vernadat (with the meaning of ``footstep").*



Point of view	*Exhaustive*	*Trace*	*Step*
# States	2^n	$(n+1)$	2
# Transition Firings	$n\times 2^n-1$	n	1

Fig. 5. Some ways to handle independent transitions, from Vernadat

The characterization given in proposition 2 can be used to build a *mathematical programming model* based on steps which can be used to solve reachability-based PN analysis problems: one has just to express the right side of equation (10) using the linear equations (8) and (9) over integer variables. Such a model will be presented in section 4.

The advantage of using steps is that they allow to *reduce* the number of firings in our model – and then the number of variables – while keeping an equivalence with the initial properties. Thus it is not a *modification of the semantics* of PNs, but only a way to *capture the independence of transitions*. Of course, this reduction does not systematically holds, since it is easy to construct a Petri net where only one transition can be fired at a time. Thus, in the worst of cases, the step firings formulation may not bring any improvement as far as the *number* of firings used are concerned. However, this is a quite uncommon situation since it means that the Petri net does not show any *parallelism*.

3.2 Incremental search

We have seen the interest of using steps to formulate the reachability problem in PNs as a search for instanciations of integer variables constrained by a system of linear equations. This formulation allows us to use the paradigm of *mathematical programming* to solve the reachability problem. However, the *initial definition* of the reachability problem is not well adapted to the kind of formulation we propose to use, since definition 5 does not make any assumption concerning the number of steps needed to solve the reachability problem. In this paragraph, we define two *sub-problems* associated with the original reachability problem introduced before, which can be conveniently solved using the characterization of proposition 2 in a mathematical programming framework.

Definition 16 (Fixed Depth Reachability Problem). *Let (R, m_0) be a Petri net, $k \in \mathrm{N}$ and m_f a marking.*

$\mathrm{P}_1(k)$	*Find a step sequence allowing to reach the marking m_f from the marking m_0 in at most k steps .*

Definition 17 (Shortest Length Reachability Problem). *Let (R, m_0) be a Petri net, and m_f a marking reachable from m_0 .*

P_2	*Find the minimal length, denoted by $K_{\min}$, of a sequence of steps allowing to reach the marking m_f from the marking m_0 .*

Of course, each of these sub-problems is *directly linked* to the initial one defined before, and each allows to solve a different kind of PN reachability analysis. For instance, the first formulation $\mathrm{P}_1(k)$ is highly useful for *model-checking* since it can serve to define an exhaustive search of the reachability graph. On the other hand, the second formulation P_2 is well designed to deal with *performance analysis* since it returns a firing sequence that *maximizes the parallelism* of the system. It can also give an helpful bound for the definition of *additional heuristics*. Finally, since it is clear that the complexity of the problem grows (w.r.t. number of variables and constraints) as the length k of the sequence of steps used increases, it seems also quite reasonable to search for the smallest value of the parameter k from which a solution exists.

The *fixed depth reachability problem* $\mathrm{P}_1(k)$ has already been studied by (Benasser, 2000) using the *logical abstraction technique*. His approach is based on the same notion of steps , but it uses constraint programming techniques. His algorithm iterates the number of steps used, adding one new step at each iteration, in order to test *all the lengths* of sequences of steps lower than k . Benasser proved that his algorithm is *correct* since the sequences found are effectively sequences of steps which produce the desired final marking. It is also *complete* since it can enumerate *all* the solutions of length smaller than a given integer k . In each iteration, the algorithm uses a mechanism of linear constraints solving. It has been implemented using the constraint logic programming software Prolog IV. The interest of using a constraint logic programming framework is that its resolution mechanism is *incremental* (Jaffar et al., 1992). Indeed, it is not necessary to redefine in each iteration the constraints incorporated into the previous stage. The constraints are added in the constraints solver so that it can reuse the results of the previous constraints propagation. The search for the concrete results is made at the end by an *enumeration* of all the possible integer solutions, which corresponds exactly to the sub-problem formulation $\mathrm{P}_1(k)$.

In section 4, we will adapt Benasser's algorithm to our own mathematical programming framework. To achieve the same kind of results, we will prove the *correctness* and *completeness* of our mathematical programming formulation with respect to $\mathrm{P}_1(k)$. These results will allow us to use integer linear programming techniques to find every solution of

$P_1(k)$. Some objective functions would also be defined to guide the search directly to an *optimal solution* in some way. Since P_2 can be easily expressed by iterating $P_1(k)$ instances for growing values of the parameter k, it will also be solved using the same technique.
Here, *Operational Research* techniques replace *Artificial Intelligence* ones, but the algorithm structure is the same. All techniques based on incremental approaches may share the same search algorithms. The most basic algorithm consists in searching in an incremental way amongst sequences the length of which are increased one by one.

3.2.1 Naive algorithm

This algorithm is fed with a *bound* K_{max} on what we call the ``*search depth*'' in order to prevent an *infinite loop*. Once chosen this value, the procedure generates iteratively a sequence of mathematical models of increasing size, and search for solutions in the corresponding search spaces using mathematical programming techniques. If there is no solution in less than k_{max} steps, the algorithm stops. It is described in Fig. 6.

```
1:  $k \leftarrow 0$
2:  **DO**
3:      $k \leftarrow k+1$
4:      Generate $\mathrm{MP}(k)$, a mathematical programming model for the problem $P_1(k)$ (which
        corresponds to characterization of proposition 2 with $k$ steps ).
5:      Solve the model $\mathrm{MP}(k)$ using branch & bound techniques (e.g. Cplex solver). Let
        $\overrightarrow{X}_{i\,[\![1,k]\!]}$ be an optimal solution of $\mathrm{MP}(k)$ if it exists.
6:      **IF** ( $\mathrm{MP}(k)$ *has a solution*), **RETURN** $\overrightarrow{X}_{i\,[\![1,k]\!]}$
7:  **WHILE** ( $\mathrm{MP}(k)$ *is infeasible*) **AND** ( $k \le K_{\max}$ )
```

Fig. 6. Naive Search Algorithm

During the formulation of the mathematical programming model at step 4, one should take care of the *domain of variables* representing the steps . Indeed: the definitions of step and step firings do not forbid *empty steps* leaving the markings unchanged. By considering empty steps valid in our formulations, we get the following result.

Proposition 3 (Satisfaction Monotony) *Let* $k \in \mathbb{N}$ *. If the problem* $P_1(k)$ *is feasible, then for any integer* $k' \ge k$ *, the problem* $P_1(k')$ *is also feasible.*

Proof. It is easy to construct a feasible solution for $P_1(k')$ from a feasible solution of $P_1(k)$ for $k' \ge k$ by adding empty steps. W

Note the same result would be true when dealing with the family of mathematical programming models $\mathrm{MP}(k)$: if there exists $k \in \mathbb{N}$ such that $\mathrm{MP}(k)$ admits a solution, any model $\mathrm{MP}(k')$ with $k' \ge k$ would be feasible too. This property motivates the jump search techniques proposed in the next paragraph.

3.2.2 Jump search

From proposition 3 and the definition of parameter $K_{\min}$, we get:

$$\begin{cases} \forall k < K_{\min} & \mathrm{P}_1(k) \text{ is infeasible} \\ \forall k \geq K_{\min} & \mathrm{P}_1(k) \text{ is feasible} \end{cases} \tag{11}$$

This property can help us to define new iterative techniques, since – for example – it shows that it is not necessary to solve all the problems $\mathrm{P}_1(k)$ for $k \leq K_{\min}$, like in the naïve search described before.
Of course, as said before, we must keep using an *incremental procedure* in order to avoid the use of large models if they are not needed. We propose finally some techniques based on *jumps* over the values of the search depth. These techniques allow to *decrease* the number of iterations needed, thus improving the search efficiency. Several jump strategies are possible. We describe briefly some elementary ones.

- **Forward jump search** The first family *continously increases* the value of the search depth. We can distinguish two main politics, depending on how the amplitude of jumps is defined.
    - **Fixed amplitude** Its value must be chosen in order to obtain a high exploration speed while minimizing the possible redundant steps. This type of strategy allows to estimate the profit precisely.
    - **Dynamic amplitude** This second strategy uses *variable amplitudes*. Increasing amplitudes should be used for small values of k, and decreasing ones when the exploration becomes more difficult. This kind of behavior is less easily quantifiable.

  These politics both can lead to overtake $K_{\min}$ when a solution is found. In this case, it is not anymore possible to answer precisely the problem P_2, since we do not get the exact value of $K_{\min}$. To compensate this lack of information, one can use a dichotomic search.
- **Dichotomic search** This kind of procedure needs to know a maximal bound for k. Its value is given by a previous successful execution of the forward jump search.

The main interest of jump search is that it allows to win in *efficiency*. Since we do not know the number of steps needed to find a solution if it exists, the use of such a technique allows us, when it is possible, not to have to develop the entire set of formulations of length lower than $K_{\min}$. Numerical experiments show that even if the *size* of models is increasing, the corresponding *practical complexity* does not always follows the same evolution.
Finally, it must be said that the procedure described in Fig. 6 is only a *semi-complete* one. Indeed, in the context of *unbounded* PNs, the value of $k_{\max}$ is set arbitrarily, as we do not know any information on the number of steps needed to find a possible solution. Thus, if *no solution* is obtained *before* the value of k has been reached, one *cannot conclude* on the reachability property.

To the contrary, when dealing with bounded PNs, it is possible to set $K_{\max}$ to the value of the *sequential depth* of the net, a parameter we have defined in (Bourdeaud'huy et al., 2004a) and which *guarantee* the *complete exploration* of the reachability graph. Using this parameter as search depth, it is *always possible* to conclude when the algorithm stops.

3.3 Adaptation to timed Petri nets

We have seen in the previous section the awaited benefits from using an incremental approach made of step firings. Before introducing the mathematical models in section 4, we propose to adapt the step based formulation to Timed Petri nets.
We start by adapting the previous formalism to Timed Petri nets. The key idea is again to consider the evolution of a Timed Petri net ``*step by step*''.

Definition 18 (Timed step). *Let* (R,d) *be a Timed Petri net. A timed step is a pair* $\psi = (\varphi, v)$ *such that:*

- $\varphi = \sum_{j \in [\![1,n]\!]} \alpha_j \cdot t_j$ *is a step* $\in \mathrm{T}^*$ *for the Place/Transition Petri net* R, *such that* $\forall j \in [\![1, N]\!], \alpha_j \in \{0,1\}$;
- v is a date $\in \mathrm{N}$.

The set of all timed steps of a Timed Petri Net is denoted by $\mathrm{T}^*_{\mathrm{TPN}}$.

Definition 19 (Timed steps Firings). *Let* (R,d) *be a Timed Petri net. Let* $e = (\overrightarrow{E_m}, \overrightarrow{E_r})$ *be a state given at date* v. *Let* $v' \geq v$ *and* $\Delta_v = v' - v \in \mathrm{N}$. *The timed step* $\psi = (\varphi, v')$ *is fireable from e iff:*

$$\forall t \in \varphi, \overrightarrow{E_r}\Big|_t \leq \Delta_v \tag{12}$$

$$C^- \cdot \overrightarrow{\varphi} \leq \overrightarrow{E_m} + C^+ \cdot \sum_{\substack{t \in \mathrm{T}, \\ 0 < \overrightarrow{E_r}|_t \leq \Delta_v}} \overrightarrow{e_t} \tag{13}$$

If this condition is satisfied, the new state $e' = (\overrightarrow{E'_m}, \overrightarrow{E'_r})$ *reached at date* v' *from* e *by the firing of* $\psi = (\varphi, v')$ *is defined as:*

$$\overrightarrow{E'_m} = \overrightarrow{E_m} - C^- \cdot \overrightarrow{\varphi} + C^+ \cdot \sum_{\substack{t \in \mathrm{T}, \\ 0 < \overrightarrow{E_r}|_t \leq \Delta_v}} \overrightarrow{e_t} \tag{14}$$

$$\forall t \in \mathrm{T}, \overrightarrow{E'_r}\Big|_t = \begin{cases} d(t) & \text{if } t \in \varphi \\ \overrightarrow{E_r}\Big|_t - \Delta_v & \text{if } \overrightarrow{E_r}\Big|_t - \Delta_v > 0 \\ 0 & otherwise \end{cases} \tag{15}$$