MODELING AND ANALYSIS OF TIMED PETRI NETS USING HEAPS OF PIECES

STÉPHANE GAUBERT† JEAN MAIRESSE‡

† INRIA, Domaine de Voluceau, B.P. 105, 78153 Le Chesnay Cedex, France.
email: Stephane.Gaubert@inria.fr, fax: +33 (0)1 39 63 57 86

‡ LIAFA, CNRS-Université Paris 7, 4 place Jussieu, 75252 Paris Cedex 05.
email: mairesse@litp.ibp.fr

Abstract

We show that safe timed Petri nets can be represented by special automata over the (max,+) semiring, which compute the height of heaps of pieces. This extends to the timed case the classical representation à la Mazurkievicz of the behavior of safe Petri nets by trace monoids and trace languages. For a subclass including all safe Free Choice Petri nets, we obtain reduced heap realizations using structural properties of the net (covering by safe state machine components). We illustrate the heap-based modeling by the typical case of safe jobshops. For a periodic schedule, we obtain a heap-based throughput formula, which is simpler to compute than its traditional timed event graph version, particularly if one is interested in the successive evaluation of a large number of possible schedules.

Keywords—Timed Petri nets, automata with multiplicities, heaps of pieces, (max,+) semiring, scheduling.

I. Introduction

Untimed and timed Petri nets have been actively studied for a long time, in particular as a modeling and analysis tool for Discrete Event Systems. Two different kinds of algebraic objects have been introduced for untimed and timed Petri nets respectively.

(1) The *untimed* behaviors of Petri nets can be represented by *languages* (sets of possible firing sequences). In the simple case of a net with bounded markings, the language of the net is rational since it is recognized by a finite automaton, the reachability graph. (2) The *timed* behavior of a net can be represented by dater functions which provide the occurrence time of all the possible events in the system. The most satisfactory results are relative to the subclass of timed Event Graphs, which can be modeled by finite dimensional recurrent linear systems over the (max,+) semiring, see [2], [9].

There exists a striking connection between the two cases: both (max,+) linear systems and conventional automata are specializations of (max,+) automata, i.e. *automata with multiplicities* [14] over the (max,+) semiring. This constatation leads to the natural question:

What is the modeling power of (max,+) automata in terms of timed Petri nets?

The purpose of this paper is to propose the following answer:

Timed safe Petri nets are special (max,+) automata, which compute the height of heaps of pieces.

Indeed, the firing times of the transitions of the net can be read on the upper contour of a Tetris-like heap of pieces, canonically associated to an (earliest) execution of the system.

This representation theorem is best understood by comparison with the following existing approaches.

1. *Trace languages.* In the landmark paper [20], Mazurkiewicz observed that *trace monoids* (that is, free partially commutative monoids) and their subsets (trace languages) are the natural model of the untimed behavior of safe Petri nets. Trace monoids, in which certain (independent) letters commute, and others do not, allow us to identify the different sequential representations of the same concurrent event.

2. *Heaps of Pieces.* In [26], Viennot observed that trace monoids are isomorphic to *heap monoids*, that is, monoids in which the generators are *pieces* (in the nearly usual understanding: these pieces are solid rectangular-shaped blocks), and where the concatenation consists in piling up one *heap* above the other. This yields a very intuitive graphical representation of trace monoids.

3. *(Max,+) linear representations*. A next step was the observation that the height of heaps of pieces is recognized by a (max,+) automaton, the result holding for general, polyomino-shaped, pieces. This was proved in Gaubert and Mairesse [16], and in a different form, by Brilman and Vincent [27], [6].

The heap representation theorem for safe timed Petri nets that we give can be seen as a synthesis of these three results. The essential idea is that considering general pieces in heap models enables to model time through the height of a piece.

Heap representations are particularly well adapted to algebraic computations. As a typical illustration, we derive an heap-based performance evaluation method for safe jobshops. The assignment of the jobs on the machines is fixed but not the order on which the jobs are processed by the machines (the schedule). The classical modeling associates with each chosen schedule an event graph (i.e. a (max,+) linear system) whose size grows with the period of the schedule, see [9], [18], [2]. On the other hand, the representation by heap model is independent of the (even non periodic) schedule which is considered. This is particularly interesting for the successive evaluation of a large number of schedules. For a periodic schedule, we propose a new heap-based algorithm to compute the throughput of the jobshop. The

traditional (event graph based) method runs in *quadratic* time, with respect to the length of the schedule period. The execution time of the heap-based algorithm is of the same order, except that it is only *linear* in the length of this period.

It is worth noting that heaps of pieces are essentially an extension of the Gantt charts traditionally used in scheduling. Whereas conventional Gantt charts only display the resource (machine) occupation times, heaps of pieces contain the complete time information for both resources *and* jobs, which allows us to write dynamical equations.

Let us mention some related independent work of Hulgaard [19] on the subclass of safe Free Choice Petri net. Hulgaard did not put forward the automata or heap model, but he did introduce (for time analysis purposes) dater variables and dynamical equations similar to the ones used here.

The paper is organized as follows. In §II, we introduce heap models and (max,+) automata. In §III, we recall basic facts about Petri nets and their representation by trace monoids. The main result, "safe timed Petri nets can be represented by heaps of pieces", is proved in §IV-A.

In this representation, the size of the heap model is equal to the number of places in the Petri net. In §IV-B, we show the existence of a much smaller heap representation for the subclass of nets which can be covered by safe State Machine Components. It includes in particular all safe Free Choice Petri nets.

In §V, we apply these results to the typical example of jobshops, proposing a new performance evaluation algorithm.

The proofs (which are ommited here due to the lack of space), together with additional discussions, can be found in an extended version of the paper[1].

II. Heap Models and (max,+) Automata

The following heap model generalises the heaps of pieces of Viennot [26].

Imagine an horizontal axis with a finite number of slots. A *piece* is a solid (possibly non connected) "block" occupying some of the slots, with staircase-shaped upper and lower contours, see Fig. 1. With an ordered sequence of pieces, we associate a *heap* by piling up the pieces, starting from an horizontal ground. This piling occurs in the intuitive[2] way: a piece is only subject to vertical translations and occupies the lowest possible position, provided it is above the ground and the pieces previously piled up.

Let us propose a more formal definition.

DEFINITION II.1. A *heap model* is a 5-tuple $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$, where:

- $\mathcal{T}$ is a finite set whose elements are called *pieces*.
- $\mathcal{R}$ is a finite set whose elements are called *slots*.
- $R : \mathcal{T} \to \mathcal{P}(\mathcal{R})$ gives the subset of slots occupied by a piece. We assume that each piece occupies at least one slot: $\forall a \in \mathcal{T}, R(a) \neq \emptyset$.
- $l : \mathcal{T} \times \mathcal{R} \longrightarrow \mathbb{R} \cup \{-\infty\}$ gives the height of the lower contour of the piece at the different slots. $u : \mathcal{T} \times \mathcal{R} \longrightarrow \mathbb{R} \cup \{-\infty\}$ (with $u \geq l$) gives the height of the upper contour of the piece. By convention, $l(a,r) = u(a,r) = -\infty$ if $r \notin R(a)$ and $\min_{r \in R(a)} l(a,r) = 0$.

A piece a occupies a region of the $\mathcal{R} \times \mathbb{R}^+$ plane of the form $\{(r,y) \in R(a) \times \mathbb{R}^+ \mid \lambda + l(a,r) \leq y \leq \lambda + u(a,r)\}$, where $\lambda \in \mathbb{R}^+$ depends on the pieces already piled up, and $\lambda = 0$ if there is no other piece.

We will interpret a length k word[3] $w = a_1, \ldots, a_k \in \mathcal{T}^*$ as a heap, i.e. as a sequence of k pieces $a_1, \ldots, a_k$ piled up in this logical order.

We define the *upper contour* of the heap w as the $\mathcal{R}$-dimensional row vector $x_{\mathcal{H}}(w)$, where $x_{\mathcal{H}}(w)_r$ is the height of the heap on slot r. The horizontal ground assumption yields $x_{\mathcal{H}}(\varepsilon) = (0, \cdots, 0)$. The *height* of the heap w is

$$y_{\mathcal{H}}(w) = \max_{r \in \mathcal{R}} x_{\mathcal{H}}(w)_r \ .$$

A useful interpretation of a heap model consists in viewing pieces as tasks and slots as resources. Each task a requires a subset of the resources (given by $R(a)$) during a certain amount of time ($u(a,r) - l(a,r)$ for a resource $r \in R(a)$). In the simplest case where $l(a,r) = 0, \forall r \in R(a)$, the execution of a task begins as soon as all the required resources, used by earlier tasks, become free. For more details along these lines, see [16], [6].

Borrowing the terminology of [2], [15], the maps $y_{\mathcal{H}}$ and $x_{\mathcal{H}}$ are called the *dater functions* of the heap model.

The piling mechanism and the different notations are best understood graphically and on an example, see Fig. 1.

Example II.2. Let us consider the following heap model.

- $\mathcal{T} = \{a, b, c, d\}, \mathcal{R} = \{1, 2, 3, 4\}$;
- $R(a) = \{1, 2, 3\}, R(b) = \{1, 2\}, R(c) = \{2, 4\}, R(d) = \{2, 3, 4\}$;
- $u(a, .) = [1, 1, 3, -\infty], l(a, .) = [0, 0, 0, -\infty]$,
  $u(b, .) = [3, 2, -\infty, -\infty], l(b, .) = [0, 0, -\infty, -\infty]$,
  $u(c, .) = [-\infty, 2, -\infty, 2], l(c, .) = [-\infty, 0, -\infty, 0]$,
  $u(d, .) = [-\infty, 1, 3, 1], l(d, .) = [-\infty, 0, 0, 0]$.

We have represented, in Fig. 1, the heap associated with the word $w = abcd$. Piece c is an example of a non connected (but "rigid") piece.

We can read directly on Fig. 1 the values $x_{\mathcal{H}}(w) = [4, 6, 8, 6]$ and $y_{\mathcal{H}}(w) = 8$.

DEFINITION II.3. The (max,+) semiring[4] $\mathbb{R}_{\max}$ is the set $\mathbb{R} \cup \{-\infty\}$, equipped with the operation $\max$, written additively (i.e. $a \oplus b = \max(a, b)$) and the usual sum, written multiplicatively (i.e. $a \otimes b = a + b$). In this semiring, $\mathbb{0} = -\infty$, $\mathbb{1} = 0$.

Note that $\mathbb{R}_{\max}$ is a dioid, i.e. a semiring with an idempotent addition ($a \oplus a = a$). We shall use throughout the paper the

[1] Accessible from the authors' Web pages http://amadeus.inria.fr/gaubert, and http://www-litp.ibp.fr/~mairesse/.

[2] It corresponds for example to the mechanism of the Tetris game.

[3] We recall the following standard notations. Given a finite set (alphabet) $\mathcal{T}$, we denote by $\mathcal{T}^n$ the set of words of length n on $\mathcal{T}$. We denote by $\mathcal{T}^* = \cup_{n \in \mathbb{N}} \mathcal{T}^n$ the free monoid on $\mathcal{T}$, that is, the set of finite words equipped with concatenation. The unit (empty word) will be denoted ε. The length of the word w will be denoted $|w|$. We shall write $|w|_a$ for the number of occurrences of a given letter a in w.

[4] A set K equipped with two operations $\oplus$ and $\otimes$ is a semiring if $\oplus$ is associative and commutative, $\otimes$ is associative and distributive with respect to $\oplus$, there is a zero element $\mathbb{0}$ ($a \oplus \mathbb{0} = a, a \otimes \mathbb{0} = \mathbb{0} \otimes a = \mathbb{0}$) and a unit element $\mathbb{1}$ ($a \otimes \mathbb{1} = \mathbb{1} \otimes a = a$).

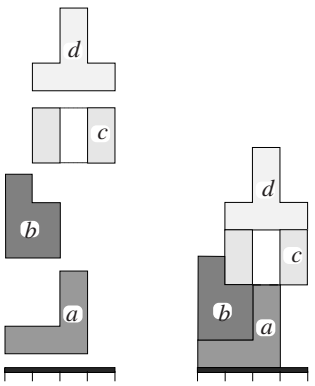


Fig. 1. Heap of pieces associated with the word $w = abcd$.

matrix and vector operations induced by the semiring structure. For matrices A, B of appropriate sizes, $(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} = \max(A_{ij}, B_{ij})$, $(A \otimes B)_{ij} = \bigoplus_k A_{ik} \otimes B_{kj} = \max_k (A_{ik} + B_{kj})$, and for a scalar a, $(a \otimes A)_{ij} = a \otimes A_{ij} = a + A_{ij}$. We will omit the $\otimes$ sign, writing for instance AB instead of $A \otimes B$ as usual. Given a set $\mathcal{S}$, we denote by $\mathbb{1}_{\mathcal{S}}$ the $|\mathcal{S}|$-dimensional column vector whose entries are all equal to $\mathbb{1}$.

DEFINITION II.4. Given a finite alphabet $\mathcal{T}$, a (max,+) automaton[5] is a 4-tuple $\mathcal{A} = (Q, I, F, \mathcal{M})$, where

- Q is a finite set (of *states*);
- $I \in \mathbb{R}_{\max}^{1\times Q}$ and $F \in \mathbb{R}_{\max}^{Q\times 1}$ are the *initial* and *final* vectors, respectively;
- $\mathcal{M}$ is a morphism $\mathcal{T}^* \to \mathbb{R}_{\max}^{Q\times Q}$.

The morphism $\mathcal{M}$ is uniquely specified by the finite family of $Q \times Q$ matrices, $\mathcal{M}(a), a \in \mathcal{T}$. Then, for a word $w = a_1 \dots a_n$, we have

$$\mathcal{M}(w) = \mathcal{M}(a_1 \dots a_n) = \mathcal{M}(a_1) \dots \mathcal{M}(a_n) \ , \qquad (1)$$

the matrix product being interpreted in the (max,+) semiring.

Let us define the vectors $x_{\mathcal{A}}(w) = I\mathcal{M}(w) \in \mathbb{R}_{\max}^{1\times Q}$ and the scalars $y_{\mathcal{A}}(w) = I\mathcal{M}(w)F \in \mathbb{R}_{\max}$ associated with the (max,+) automaton $\mathcal{A}$. We say that $x_{\mathcal{A}}$ and $y_{\mathcal{A}}$ are *recognized*[6] by the automaton $\mathcal{A}$.

We have

$$\begin{aligned} x_{\mathcal{A}}(e) &= I \ , \\ x_{\mathcal{A}}(wa) &= x_{\mathcal{A}}(w)\mathcal{M}(a) \ , \\ y_{\mathcal{A}}(w) &= x_{\mathcal{A}}(w)F \ . \end{aligned} \qquad (2)$$

Hence a (max,+) automaton may be seen as a (max,+) linear system whose dynamic is driven by letters. With a heap model $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$, we associate the morphism $\mathcal{M} : \mathcal{T}^* \to \mathbb{R}_{\max}^{\mathcal{R}\times\mathcal{R}}$, given by

$$\mathcal{M}(a)_{sr} = \begin{cases} \mathbb{1} & \text{if } s = r, r \notin R(a), \\ u(a,r) - l(a,s) & \text{if } r \in R(a), s \in R(a), \\ \mathbb{0} & \text{otherwise.} \end{cases} \qquad (3)$$

[5]This is the specialization to the (max,+) semiring of the classical notion of automaton with multiplicity [14], or recognizable series [25], [4].

[6]Classically, in automata theory, only the map $y_{\mathcal{A}}$ is said to be recognized. It is convenient here to extend the definition of recognizability.

THEOREM II.5. *Let $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$ be a heap model. The (max,+) automaton $(\mathcal{R}, \mathbb{1}_{\mathcal{R}}^t, \mathbb{1}_{\mathcal{R}}, \mathcal{M})$ recognizes the upper contour $x_{\mathcal{H}}$ and the height $y_{\mathcal{H}}$:*

$$\forall w \in \mathcal{T}^*, \qquad \begin{aligned} x_{\mathcal{H}}(w) &= \mathbb{1}_{\mathcal{R}}^t \mathcal{M}(w) \ , \\ y_{\mathcal{H}}(w) &= \mathbb{1}_{\mathcal{R}}^t \mathcal{M}(w) \mathbb{1}_{\mathcal{R}} \ . \end{aligned}$$

A variant of this result was proved in [16], see also [6]. We will call $(\mathcal{R}, \mathbb{1}_{\mathcal{R}}^t, \mathbb{1}_{\mathcal{R}}, \mathcal{M})$ the heap automaton representing the heap model.

Example II.6. Let $\mathcal{T} = \{a, b, c, d\}$, $Q = \{1, 2, 3, 4\}$, $I = \mathbb{1}_Q^t$, $F = \mathbb{1}_Q$.

$$\mathcal{M}(a) = \begin{bmatrix} 1 & 1 & 3 & \\ 1 & 1 & 3 & \\ 1 & 1 & 3 & \\ & & & \mathbb{1} \end{bmatrix}, \ \mathcal{M}(b) = \begin{bmatrix} 3 & 2 & & \\ 3 & 2 & & \\ & & \mathbb{1} & \\ & & & \mathbb{1} \end{bmatrix},$$

$$\mathcal{M}(c) = \begin{bmatrix} \mathbb{1} & & & \\ & 2 & & 2 \\ & & \mathbb{1} & \\ & 2 & & 2 \end{bmatrix}, \ \mathcal{M}(d) = \begin{bmatrix} \mathbb{1} & & & \\ & 1 & 3 & 1 \\ & 1 & 3 & 1 \\ & 1 & 3 & 1 \end{bmatrix}.$$

(the $\mathbb{0}$ entries are omitted).

One verifies easily that the (max,+) automaton $(Q, I, F, \mathcal{M})$ represents the heap model given in Ex. II.2. We have

$$\mathcal{M}(abcd) = \begin{bmatrix} 4 & 6 & 8 & 6 \\ 4 & 6 & 8 & 6 \\ 4 & 6 & 8 & 6 \\ \mathbb{0} & 3 & 5 & 3 \end{bmatrix},$$

$$x_{\mathcal{A}}(abcd) = I\mathcal{M}(abcd) = \begin{bmatrix} 4 & 6 & 8 & 6 \end{bmatrix}, y_{\mathcal{A}}(abcd) = 8 \ .$$

This provides an algebraic confirmation of the values obtained graphically in Fig 1.

III. Untimed and Timed Petri Nets

A. Definitions

We next recall some classical facts about untimed and timed Petri nets, needed to specify the subclass of timed Petri nets that we will be able to deal with. For more details on Petri nets, see Murata [22], Desel and Esparza [11] or the proceedings [5].

DEFINITION III.1. A Petri net (PN) is a 4-tuple $(\mathcal{G}, M) = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M)$, where :

- $\mathcal{P}$ is a finite set, whose elements are called *places*.
- $\mathcal{T}$ is a finite set, whose elements are called *transitions*.
- $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a relation between places and transitions.
- M is a map $\mathcal{P} \to \mathbb{N}$. The integer $M(p)$ is called the *initial marking* of place p.

A Petri net $\mathcal{G}$ is traditionally represented as a bipartite directed graph. There are two different kinds of nodes, places $p \in \mathcal{P}$, (represented by circles) and transitions $a \in \mathcal{T}$ (represented by rectangles). An element of $\mathcal{F}$ is an arc from a place to a transition or from a transition to a place. It is therefore natural to speak of "input places", "output transitions" and so on. We use the notations ${}^\bullet p, p^\bullet$ (resp. ${}^\bullet a, a^\bullet$) for the set of input and output

transitions of place p (resp. input and output places of transition a). The marking $M(p)$ is displayed by drawing $M(p)$ tokens in place p.

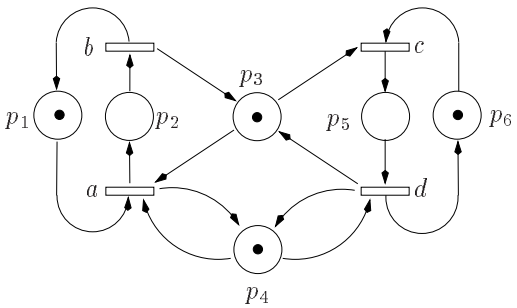


Fig. 2. Strongly connected safe Petri net

Without loss of generality, we consider only connected Petri nets (in the sequel, this assumption is made without further notice). Indeed, if a net is not connected, we can consider each connected component independently.

A Petri net is a dynamic object. The underlying structure $(\mathcal{P}, \mathcal{T}, \mathcal{F})$ is never modified, but the marking M evolves according to the following *firing rule*.

1. Transition a is *enabled* at M if there is at least one token in each of its input places.
2. An enabled transition a can *fire*. The firing of a transforms M into M' (written $M \xrightarrow{a} M'$) by removing one token from each of the input places and adding one token in each of the output places of a.

We say that a word $w = a_1 a_2 \cdots a_n \in \mathcal{T}^*$ is a *firing sequence* starting from marking M' if there is a sequence of markings $M' = M_0, M_1, \ldots, M_n = M''$ such that transition a_i is enabled at M_{i-1} and $M_{i-1} \xrightarrow{a_i} M_i$. We abbreviate this by $M' \xrightarrow{w} M''$. A marking M'' is *reachable* from a marking M' if there is a firing sequence $w \in \mathcal{T}^*$ such that $M' \xrightarrow{w} M''$. We denote by $\mathcal{R}(M')$ the set of markings reachable from M'.

We call *language*[7] *of the Petri net* $(\mathcal{G}, M)$ the set $L \subset \mathcal{T}^*$ of firing sequences starting from M.

DEFINITION III.2. A Petri net $(\mathcal{G}, M)$ with language L is:

- *live* if $\forall w \in L, \forall t \in \mathcal{T}, \exists u \in \mathcal{T}^*$, such that $wut \in L$, i.e. if whatever the past firings (= w) are, it is possible to find a firing sequence from the current state, containing transition t;
- *bounded* if the set $\mathcal{R}(M)$ is finite. Equivalently, if $\exists k$ such that $\forall M' \in \mathcal{R}(M), \forall p \in \mathcal{P},\ M'(p) \leq k$;
- *safe* (or 1-bounded) if a place will not hold more than one token: $\forall M' \in \mathcal{R}(M), \forall p \in \mathcal{P},\ M'(p) \leq 1$.

Example III.3. The Petri net represented in Fig. 2 is live and safe. Its language is $L = (ab \cup cd)^*(e \cup a \cup c)$.

[7] More properly, the **P**-type free labeled language, according to the terminology of Peterson [23]. Various kinds of Petri net languages have been defined (according to various kinds of acceptance conditions on final markings and the different labeling functions that one may consider).

B. Execution semantics and traces

Let us consider a Petri net where n transitions, a_1 to a_n, are concurrently enabled (several transitions are concurrently enabled whenever they can be fired independently). We assume that all of them have to fire before any new transition becomes enabled. Then, the same behavior, i.e. the firing of the n transitions, can be described by any of the following $n!$ firing sequences: $a_{\sigma(1)}, \cdots, a_{\sigma(n)}$ where σ is a permutation of $\{1, \ldots, n\}$.

This simple example shows that firing sequences, which provide a sequential description of the behavior, are not really adapted in the presence of concurrency.

The problem of modeling concurrency in a more efficient way has long been considered. A classical approach, proposed by Mazurkiewicz [20], [21], uses the notion of trace monoid. For a general and recent reference on traces, see [13].

DEFINITION III.4. Let $\mathcal{T}$ be an alphabet equipped with a reflexive symmetric relation called *dependence relation* and denoted by D. We denote by I the complement of D, called *independence* relation. The *trace monoid* $(\mathcal{T}^*/_\sim)$ is the quotient of the free monoid $\mathcal{T}^*$ by the least congruence $\sim$ containing the relations $ab \sim ba,\ \forall (a,b) \in I$. The elements of $(\mathcal{T}^*/_\sim)$ will be called *traces*.

Two words are representatives of the same trace if they can be obtained one from the other by repeatedly interchanging adjacent independent letters.

Given a Petri net $(\mathcal{P}, \mathcal{T}, \mathcal{F}, M)$, we define the following independence relation:

$$I = \{(a,b) \in \mathcal{T}^2 \mid ({}^\bullet a \cup a^\bullet) \cap ({}^\bullet b \cup b^\bullet) = \emptyset\} \quad , \qquad (4)$$

and the associated trace monoid $(\mathcal{T}^*/_\sim)$. Two transitions are independent iff they do not share input or output places.

For safe Petri nets[8], the congruence $\sim$ generated by I identifies firing sequences which differ only by the sequential ordering of concurrent events. In particular, if $w_1 \sim w_2$ then $w_1 \in L \Leftrightarrow w_2 \in L$ (where L is the language of the Petri net). A trace whose representatives are firing sequences is called a *firing trace*. The set of firing traces is denoted by $(L/_\sim)$.

It is very convenient to visualize trace monoids using heap models, as it was originally proposed by Viennot in [26]. Let us detail this for the trace monoid $(\mathcal{T}^*/_\sim)$ associated with a safe Petri net $(\mathcal{P}, \mathcal{T}, \mathcal{F}, M)$. Consider the heap model $\mathcal{H}(\mathcal{T}^*/_\sim)$ with:

- set of pieces $\mathcal{T}$;
- set of slots $\mathcal{P}$;
- $R(a) = {}^\bullet a \cup a^\bullet,\ a \in \mathcal{T}$;
- $l(a,r) = 0, u(a,r) = 1, r \in R(a)$.

Two words $w_1, w_2 \in \mathcal{T}^*$ are equivalent ($w_1 \sim w_2$) if and only if they provide the same heap. In the heap associated with $w \in L$, the pieces associated with one level (i.e. the pieces occupying

[8] In a non-safe Petri net, two transitions a, b such that ${}^\bullet a \cap {}^\bullet b \neq \emptyset$ can be concurrently enabled if their common input places hold more than one token. It is possible to introduce another dependence relation to take care of this, see Diekert [12]. Such refinements are not needed here as we consider only safe Petri nets.

a common vertical position) correspond to the events (i.e. the transition firings) occurring concurrently.

Example III.5. Let us consider the Petri net of Fig. 2. We have represented in Fig. 3 the heap of pieces associated with the firing sequence $abcdabcd$. Note that there is no concurrency at all in this Petri net. It is purely sequential and the independence relation I defined in (4) is empty.

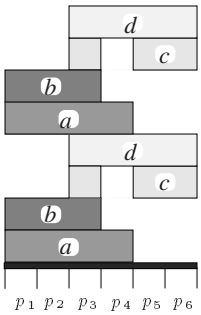


Fig. 3. Heap of pieces for the word $w = abcdabcd$.

C. Timing

A timed Petri net (TPN) is a net with *firing times* associated with transitions and/or *holding times* associated with places. Several firing semantics have been considered in the literature. We restrict our attention to safe Petri nets and consider a semantic which coincides with the one of Ramchandani [24].

Let a be a transition with firing time τ_a and whose output places $p \in a^\bullet$ have holding times τ_p. We assume that transition a becomes enabled at instant t. A firing occurs in three steps.

1. At instant t, the firing of a may be *initiated*. If initiated, it removes one token from each input place.
2. One token is added in each of the output place at instant $t + \tau_a$.
3. The token added in place $p \in a^\bullet$ can contribute to the enabling of the transitions in $p^\bullet$ after instant $t + \tau_a + \tau_p$.

Between t and $t + \tau_a$, the tokens can be considered as being 'frozen' in their original input place. The tokens and the transition a can not be involved in any other firing between t and $t + \tau_a$. A natural question to ask is what happens if transition a is not initiated at instant t. Then, either a never fires or it eventually gets disabled (because of the safeness property, this happens precisely when the token of one of the places in ${}^\bullet a$ participates in the firing of another transition). If transition a gets re-enabled later on, we are back to the original situation.

Let us investigate some other consequences of this semantic. First of all, if a transition fires, it does so "as soon as possible". We say that the Petri net operates with an *earliest firing rule*. Second, the decisions about which transitions are to fire is not based on time considerations. All logically feasible choices can be considered. This contrasts with several models studied in the literature. For example, in the so-called *race policy*, see for instance [1], a place with several output transitions allocates its token to the transition which is able to complete its firing first.

We denote by $(\mathcal{G}, M, \tau)$ a timed Petri net, where τ is a map $\mathcal{T} \cup \mathcal{P} \to \mathbb{R}^+$, providing the firing and holding times of transitions and places. By convention, the timed evolution of the Petri net starts at instant 0, in marking M, the holding times of the initial tokens being completed.

Example III.6. Let us consider the Petri net of Fig. 2. We associate with its transitions, the following firing times:

$$\tau_a = 1, \tau_b = 2, \tau_c = 2, \tau_d = 1 \ .$$

We associate with the places the holding times:

$$\tau_1 = 1, \tau_2 = 0, \tau_3 = 0, \tau_4 = 2, \tau_5 = 0, \tau_6 = 0 \ .$$

Let us assume that the initial marking is the one shown in Fig. 2. Transitions a and c are enabled. If we choose to fire transition c, the firing will be initiated at time 0, completed at time 2, and transition d will become enabled at time 2.

IV. Heap Representation for Safe Timed Petri Nets

A. Heap Representation Theorem

In this section, we state the main representation theorem of the paper: *firing times of safe timed Petri nets are recognized by heap automata*.

Let $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M, \tau)$ denote a safe timed Petri net, with set of firing sequences $L \subset \mathcal{T}^*$. The timed behavior of the net is defined as follows: for a firing sequence $w = a_1 \dots a_k \in L$, we start at time 0, and fire the transitions $a_1, \dots, a_k$ in this order, applying the earliest firing semantics described in §III-C above.

With each place p, we associate the real nonnegative numbers:

$z(w)_p$ = *instant at which the last token arrived in place p under the schedule w becomes available for the firing of downstream transitions.*

$z'(w)_p$ = *last instant of presence of a token in place p, under the schedule w.*

We set $z'(w)_p = z(w)_p = 0$, if no token was ever present in place p. We set

$$x_{\mathcal{G}}(w)_p = \begin{cases} z(w)_p & \text{if } M \xrightarrow{w} M', \text{with } M'(p) = 1 \\ z'(w)_p & \text{if } M \xrightarrow{w} M', \text{with } M'(p) = 0 \end{cases} \tag{5}$$

In words, this is the completion time of the last "event" at place p, under schedule w, an "event" being either the availability, or the departure of a token.

We call $x_{\mathcal{G}}$ the *dater vector* of the Petri net. We extend $x_{\mathcal{G}}$ to $\mathcal{T}^*$ by setting $x_{\mathcal{G}}(w)_p = \mathbb{0}$, for $w \notin L$.

The *makespan* or *execution time* of the firing sequence w is naturally defined by

$$y_{\mathcal{G}}(w) = \max_p x_{\mathcal{G}}(w)_p \ . \tag{6}$$

In words, this is the completion time of the last "event" in the Petri net under schedule w.

THEOREM IV.1 (HEAP REPRESENTATION FOR SAFE TPN). *Let $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M, \tau)$ be a safe timed Petri net. Then, the heap model $\mathcal{H} = (\mathcal{T}, \mathcal{P}, R, l, u)$, with*

$$\begin{aligned} \forall a \in \mathcal{T}, \quad & R(a) &=& \ a^\bullet \cup {}^\bullet a \ , \\ \forall a \in \mathcal{T}, \forall p \in a^\bullet, \quad & u(a,p) &=& \ \tau_a + \tau_p \ , \\ \forall a \in \mathcal{T}, \forall p \in {}^\bullet a \setminus a^\bullet, \quad & u(a,p) &=& \ 0 \ , \\ \forall a \in \mathcal{T}, \forall p \in R(a), \quad & l(a,p) &=& \ 0 \ , \end{aligned}$$

is such that

$$\forall w \in L, \qquad x_{\mathcal{G}}(w) = x_{\mathcal{H}}(w),\ y_{\mathcal{G}}(w) = y_{\mathcal{H}}(w)\ . \quad (7)$$

Equation (7) states that the dater vector of the net coincides with the upper contour of the associated heap.

Let us consider the heap model obtained from $\mathcal{H}$ by replacing u and l by $\tilde{u}(a) = 1, a \in R(a)$ and $\tilde{l}(a) = 0, a \in R(a)$. This is precisely the heap model associated with the trace monoid $(\mathcal{T}^*/_\sim)$ of the Petri net, i.e. $\mathcal{H}(\mathcal{T}^*/_\sim)$, see §III-B. The pieces of the heap model $\mathcal{H}$ are obtained by a deformation of the pieces of the heap model $\mathcal{H}(\mathcal{T}^*/_\sim)$, incorporating the timing information.

Note that for $w \notin L$, the heap associated with w and the daters $x_{\mathcal{H}}(w)$ and $y_{\mathcal{H}}(w)$ are still defined. However, they have no meaning in terms of the Petri net.

Example IV.2. Let us illustrate the previous construction with the Petri net $(\mathcal{G}, M) = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M)$ represented in Fig. 2 and the numerical values of Ex. III.6.

The heap model associated with $(\mathcal{G}, M)$ is $\mathcal{H} = (\mathcal{T}, \mathcal{P}, R, l, u)$ with

$$\begin{aligned} R(a) &= {}^\bullet a \cup a^\bullet = \{p_1, p_2, p_3, p_4\}, R(b) = \{p_1, p_2, p_3\}, \\ R(c) &= \{p_3, p_5, p_6\}, R(d) = \{p_3, p_4, p_5, p_6\}\ . \\ u(a, .) &= [0, 1, 0, 3, \mathbb{0}, \mathbb{0}], u(b, .) = [3, 0, 2, \mathbb{0}, \mathbb{0}, \mathbb{0}], \\ u(c, .) &= [\mathbb{0}, \mathbb{0}, 0, \mathbb{0}, 2, 0], u(d, .) = [\mathbb{0}, \mathbb{0}, 1, 3, 0, 1]\ . \end{aligned}$$

We have represented the heap of pieces associated with the schedule $w = abcdabcd$ in Fig. 4. For the clarity of the figure, pieces with parts of zero width have been materialized by replacing zero by a 'small' but strictly positive height. This heap is a deformation of the one of Fig. 3. The reader could check directly (by simulation of the Petri net) that the height of the heap, $y_{\mathcal{G}}(w) = 16$, attained at slot 4, corresponds to the last occurrence of an event in the system (the availability of the token in p_4, after the firing of the last occurrence of d in the schedule w).

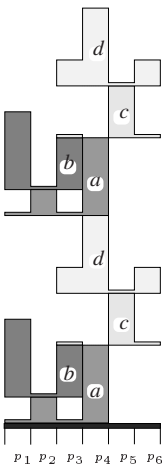


Fig. 4. Heap of pieces associated with the word $w = abcdabcd$.

B. The Minimal Realization Problem for Heaps of Pieces

The size of the heap representation in Theorem IV.1 is equal to the number of places of the Petri net, a possibly large number. This raises naturally the following *minimal realization* problem: *what is the minimal size of a heap representation of a given safe timed Petri net ?*

As a partial answer to this probably difficult problem, we will show that simple (usually small) heap representations can be built from *structural invariants*, for a subclass of nets.

To formalize this rigorously, we introduce the following definition.

DEFINITION IV.3 (HEAP REALIZATION). We say that a timed Petri net $(\mathcal{G}, M)$ with language L, has a *Heap Realization of size* k if there is a heap model $\mathcal{H}$ with k slots, such that

$$\forall w \in L,\ \ y_{\mathcal{G}}(w) = y_{\mathcal{H}}(w)\ . \quad (8)$$

That is, the execution time of the firing sequence w coincides with the height of the heap of pieces w. It is not required that $x_{\mathcal{H}} = x_{\mathcal{G}}$, which gives the potential for a smaller heap realization.

The following notions are classical, see [11, §5.1].

DEFINITION IV.4 (STATE MACHINE COVERING). A state machine is a Petri net such that each transition has exactly one predecessor and one successor. A *state machine component* of a Petri net $\mathcal{G}$ is a subnet $\mathcal{G}'$ of $\mathcal{G}$, that is a state machine, and satisfies

$${}^\bullet p \cup p^\bullet \subset \mathcal{G}' \ \text{ for every place } p \text{ of } \mathcal{G}'.$$

We say that $\mathcal{G}_1, \ldots, \mathcal{G}_k$ is a (cardinality k) *state-machine covering* of a Petri net $\mathcal{G}$, if

1. $\mathcal{G}_1, \ldots, \mathcal{G}_k$ are state-machine components of $\mathcal{G}$;
2. every arc of $\mathcal{G}$ belongs to at least one component $\mathcal{G}_i$, $1 \leq i \leq k$.

We say that the covering is *safe* (resp. *live*) if it is composed of safe (resp. live) state-machine components.

THEOREM IV.5 (REDUCED REALIZATION THEOREM). *A safe timed Petri net with a cardinality k safe state-machine covering admits a heap realization of size k.*

A heap realization of size k can be easily built by taking as resources the tokens of the state machine components. Formally, take the heap model $\mathcal{H} = (\mathcal{T}, \{1, \ldots, k\}, R, l, u)$, with

$$\begin{aligned} \forall a \in \mathcal{T},\ R(a) &= \{i \mid\ a \in \mathcal{G}_i\}\ , & (9) \\ \forall a \in \mathcal{T}, \forall i \in R(a),\ l(a, i) &= 0\ , & (10) \\ \forall a \in \mathcal{T}, \forall i \in R(a),\ u(a, i) &= \tau_a + \tau_{p(a,i)}\ , & (11) \end{aligned}$$

where $p(a, i)$ is the unique place such that the arc $(a, p(a, i))$ belongs to $\mathcal{G}_i$. A classical result (see for example [11, Th. 5.6]), which states that a live and safe free choice net admits a live and safe state machine covering, allows us to apply the reduced realization Theorem IV.5 to all live and safe free choice nets. In the special case of an event graph, state machine coverings are replaced by circuit coverings. The problem of finding a safe covering of minimal cardinality appears to be a difficult one.

Example IV.6. Let us illustrate Theorem IV.5. We consider the Petri net $(\mathcal{G}, M) = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M)$ of Fig. 2, with the timings defined in Ex. III.6.

This net admits a decomposition into 4 SM-components, $\mathcal{G}_i, i = 1, \ldots, 4$, with respective sets of places:

$$\mathcal{P}_1 = \{p_1, p_2\}, \mathcal{P}_2 = \{p_2, p_3, p_5\}, \mathcal{P}_3 = \{p_4\}, \mathcal{P}_4 = \{p_5, p_6\} .$$

Let us consider the associated heap model $\mathcal{H}$ as in Theorem IV.5. It is exactly the heap model defined in Ex. II.2 and Ex. II.6. The set of slots is $\mathcal{R} = \{1, 2, 3, 4\}$, slot i corresponding to the SM-component $\mathcal{G}_i$. The heap associated with the schedule $abcd$ was represented in Fig. 1. As a further illustration, the heaps associated with $abcdabcd$ and $abcdcdab$ are represented in Fig. 5,(1,2).

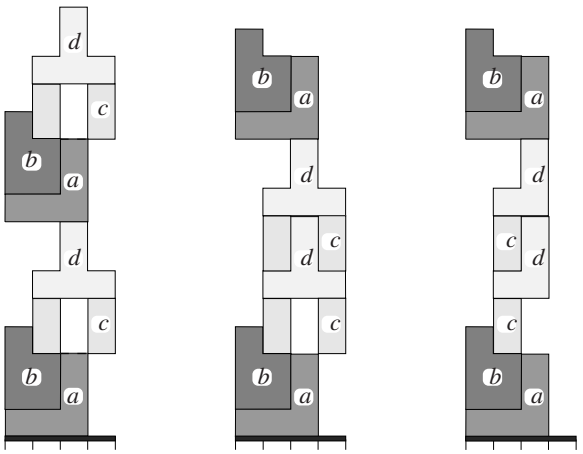


Fig. 5. (1,2): Heaps of pieces for the words $abcdabcd$ and $abcdcdab$. (3): Minimal Heap realization

It is interesting to note that the heap realization given above is not minimal. A smaller realization is shown on Fig. 5,(3). Note that this size 3 realization is not associated with a state machine covering of the net (here, the cardinality of a state machine covering is at least equal to 4).

V. An Application to the Modeling and Performance Analysis of Jobshops

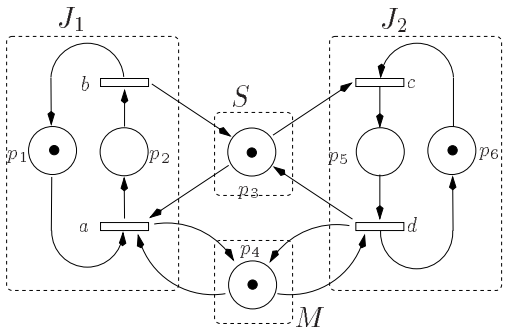
The results presented above find a natural domain of application in scheduling theory. Heap representations can be used to design efficient schedule evaluation methods. We explain informally the method on a small manufacturing model, and we compare it with the classical approach. Then, we consider the general subclass of safe jobshops. We describe a new, heap automata based, performance evaluation algorithm, that we compare with the classical performance evaluation methods.

A. An Elementary Example

Let us consider the Petri net of Fig. 6, that the reader certainly recognizes as being the one discussed extensively above.

This Petri net can be interpreted as a manufacturing system processing two job types J_1, J_2, using two (heterogeneous) resources: one specialist S and one machine M.

There are four elementary tasks a, b, c, d. The production sequence for job J_1 is ab which means that the elementary tasks a and b have to be performed in this order to complete one job J_1. The production sequence for job J_2 is cd.



Holding and Firing Times

a	b	c	d	p_1	p_2	p_3	p_4	p_5	p_6
1	2	2	1	1	0	0	2	0	0

Fig. 6. A two jobs — two resources manufacturing system

Let $w \in \mathcal{T}^*$. For each job type $J_i, i = 1, 2$, we set

$$|w|_{J_i} = \text{number of type } J_i \text{ jobs completed under the schedule } w. \quad (12)$$

Then, given an infinite schedule[9] $z = a_1 a_2 a_3 \ldots \in \mathcal{T}^\omega$, with $a_1, a_2, \ldots \in \mathcal{T}$, we may define the *asymptotic throughput* of the jobs of type J_i:

$$\lambda_i = \liminf_{n \to \infty} \frac{|a_1 \ldots a_n|_{J_i}}{\text{execution time of } a_1 \ldots a_n} . \quad (13)$$

We are interested in finding the infinite schedules maximizing the throughput, under a production ratio constraint (e.g. one job J_1 for one job J_2, in the average). We restrain this problem, by requiring the schedules to be periodic. That is, one only considers periodic sequences of the form $v^\omega = vvvv \ldots$, where v is a finite production pattern satisfying the ratio constraint. In this case, as detailed below, the $\liminf$ in (13) becomes a limit (this will follow from the (max,+) linear representation, together with the (max,+) cyclicity theorem).

For instance, let us consider minimal length patterns with ratio $1/1$. There are two possible forms for such patterns: $abcd$ and $cdab$. Moreover, we note that the asymptotic performance is invariant by cyclic conjugacy of the pattern. That is, for all words u, v, $(uv)^\omega$ and $(vu)^\omega$ have the same asymptotic throughput, which follows from the identities $(uv)^\omega = u(vu)^\omega$, $(vu)^\omega = v(uv)^\omega$ (the two behaviors differ only by a finite number of tasks). Hence, there is only one behavior to consider:

$$L_1 = (abcd)^\omega .$$

One might of course consider longer patterns. E.g. periodic sequences whose pattern consists in the production of 2 jobs J_1 and 2 jobs J_2 are given by:

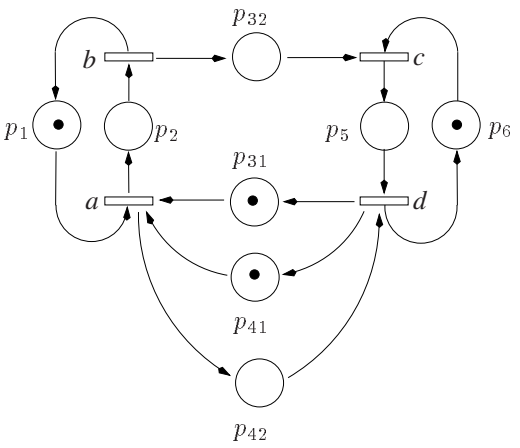
$$L_2 = (abcdabcd)^\omega \cup (abcdcdab)^\omega .$$

We will not consider as such the schedule optimization problem (which is a difficult combinatorial one), but we will show how the heap-based modeling makes easier the *subproblem* of the performance evaluation of a given periodic schedule. This is best understood by comparison with the timed Event Graph modeling, that we next recall.

[9] We denote by $\mathcal{T}^\omega$ the set of infinite words over the alphabet $\mathcal{T}$.

A.1 Illustrating the classical approach

For a given periodic schedule, one is able to build a timed Event Graph representing the system, and then to compute the periodic throughput of this timed Event Graph. Let us consider for example the schedule $(abcd)^\omega$. This functioning is represented by the timed Event Graph displayed in Fig. 7, which is obtained from the timed Petri net of Fig. 2 by replacing the resource places p_3 and p_4 by *circuits*, forcing the periodic sequence $abcdabcd\ldots$



Holding and Firing Times:

a	b	c	d	p_1	p_2	p_{31}	p_{32}	p_{41}	p_{42}	p_5	p_6
1	2	2	1	1	0	0	0	2	2	0	0

Fig. 7. Timed Event Graph for the schedule $(abcd)^\omega$

Let $x(n) \in \mathbb{R}^{\mathcal{T}}_{\max}$ be the vector providing the dates of the completion of the n-th firing of the transitions. The vector $x(n)$ evolves according to a (max,+) linear dynamic:

$$\begin{aligned} x_a(n) &= \tau_a(\tau_1 x_b(n-1) \oplus (\tau_{31} \oplus \tau_{41}) x_d(n-1)) \\ x_b(n) &= \tau_b \tau_2 x_a(n) \\ x_c(n) &= \tau_c(\tau_{32} x_b(n) \oplus \tau_6 x_d(n-1)) \\ x_d(n) &= \tau_d(\tau_5 x_c(n) \oplus \tau_{42} x_a(n)) \ . \end{aligned}$$

Setting $\xi(n) = [x_b(n), x_d(n)]^t$, eliminating x_a and x_c, and taking the numerical values of Fig. 7, we obtain the subsystem $\xi(n) = \overline{A}\xi(n-1)$, with

$$\overline{A} = \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \ , \quad \rho(\overline{A}) = 8 \ , \tag{15}$$

where $\rho(\overline{A})$ denotes the (unique) (max,+) eigenvalue of the (irreducible) matrix $\overline{A}$ (see [2], [9], particularly the introductive section §1.3 of [2], for details on the (max,+) spectral theory and its applications to discrete event systems). It follows from the cyclicity theorem in [2] that for $i = 1, 2$, and $u \in \mathcal{T}$, the asymptotic throughputs are

$$\lambda_i = \lim_{n\to\infty} \frac{n}{x(n)_u} = \frac{1}{\rho(\overline{A})} = \frac{1}{8} \ . \tag{16}$$

For a schedule with a longer period, one would have to perform a similar analysis on a larger TEG. For instance, the TEG corresponding to the schedule $w = (abcdcdab)^\omega$ is shown on Fig. 8.

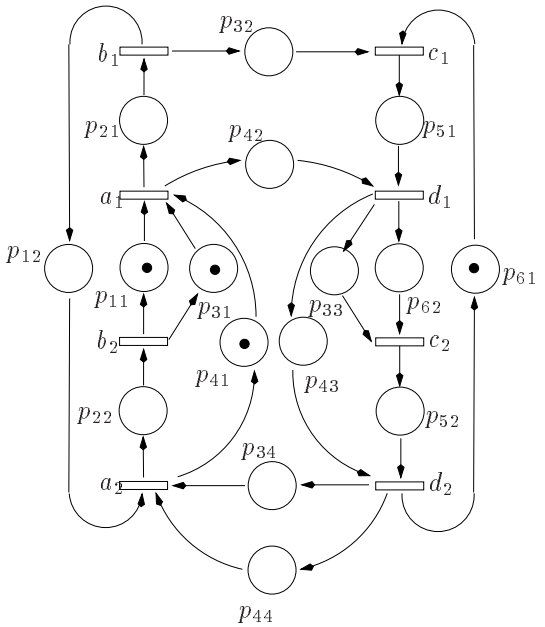


Fig. 8. Timed Event Graph for the schedule $(abcdcdab)^\omega$

A.2 Illustrating the automata approach

We associate with the timed Petri net of Fig. 6 the reduced heap model and automaton given in Ex. IV.6 above.

As opposed to what was done in the classical approach, one considers a single Petri net (the original one, Fig. 6) and a single algebraic representation (the heap model). Only the order in which the products of the matrices $\mathcal{M}(u), u \in \mathcal{T}$, are performed is modified from one schedule to the other.

In particular, for a periodic schedule $(v)^\omega$, the asymptotic throughput of job J_i is given by

$$\lambda_i = \frac{|v|_{J_i}}{\rho(\mathcal{M}(v))} \ . \tag{17}$$

For instance, the matrix $\mathcal{M}(abcd)$ was given in Ex. II.6. Its eigenvalue is $\rho(\mathcal{M}(abcd)) = 8$, providing a throughput $\lambda_i = 1/8$, which confirms the value obtained in Eqn 16.

Similarly, one may compute the matrix $\mathcal{M}(abcdcdab)$ and obtain $\rho(\mathcal{M}(abcdcdab)) = 15$, yielding a throughput $\lambda_i = 2/15 > 1/8$. This improvement of the throughput can be visualized on the heaps of pieces of Fig. 5.

These computations could be performed equivalently, and more simply, with the three dimensional matrices corresponding to the minimal realization, see Fig. 5,(3).

More generally, one can easily check that the optimal periodic schedule having a period of length $4n$ and satisfying a 1/1 ratio constraint is w_n^ω with $w_n = ab(cd)^n(ab)^{n-1}$. It can be inferred from the heaps of Fig. 5 that the associated throughput is

$$\lambda_1 = \lambda_2 = \frac{n}{(7n+1)} \ ,$$

so that λ_i increases to $1/7$ as $n \to +\infty$. This is of independent interest. It shows that we can always improve on the throughput by increasing the length of the pattern. Hence there exists no optimal schedule with a finite period (despite the fact that all durations are integer valued). An example of the same kind (but for

a non-safe Petri net) was exhibited in Carlier & Chretienne [7], §VI-1.

We next turn our attention to the general class of jobshops.

B. Performance evaluation of safe jobshops

DEFINITION V.1. A jobshop is specified by:

- a finite set $\mathcal{R}$ of resources (machines);
- a finite set $\mathcal{T}$ of elementary tasks;
- for each task $a \in \mathcal{T}$, a duration $\tau(a)$ and a single machine $R(a) \in \mathcal{R}$ on which a is to be executed;
- a finite set $\mathcal{J} \subset \mathcal{T}^*$ of production sequences or *jobs*. Each job $J = a_1 \dots a_k \in \mathcal{J}$ is composed of a finite number of tasks $a_1 \dots a_k$, to be executed in this order. We require[10] that a task $a \in \mathcal{T}$ belongs to a unique job $J(a) \in \mathcal{J}$.

We say that a unit of *job* J is produced each time the production sequence J is completed.

This model is equivalent to the one of [18]. We will make a restriction by assuming that the work in process for each job is equal to one, that is, at most one unit of each job is processed simultaneously. We call such jobshops *safe*, this assumption being equivalent to the safeness of the natural timed Petri net representation of the system, shown in Fig. 9.

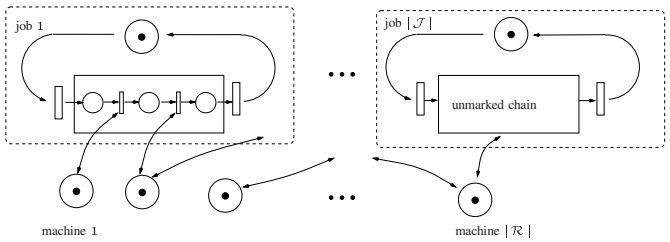


Fig. 9. Generic Petri net associated with a safe jobshop

The above Petri net admits a natural covering by live and safe state machine components, each job and machine corresponding to a component. Hence, jobshops admit reduced heap realizations (see Theorem IV.5):

$$\begin{array}{l} \mathcal{H} = (\mathcal{T}, \mathcal{R}', R', l, u),\ \mathcal{R}' = \mathcal{R} \cup \mathcal{J}, \\ R'(a) = R(a) \cup J(a), \\ l(a,r) = 0, u(a,r) = \tau(a),\ \forall r \in R(a)\ . \end{array} \tag{18}$$

We will be interested in periodic schedules of the form $v^\omega \in \mathcal{T}^\omega$, where v is a pattern, corresponding to the exact completion of several jobs (i.e. without leaving some production sequence unfinished) and meeting a fixed ratio constraint.

In line with (12), for each production sequence J, we will denote by $|v|_J$ the number of units of J completed under v.

The asymptotic throughput of job J is defined as in (13), replacing J_i by J.

THEOREM V.2 (THROUGHPUT FORMULA). *For a safe jobshop with heap realization (18) and associated matrix representation (3), the asymptotic throughput of job J is given by*

$$\lambda_J = |v|_J \times \rho(\mathcal{M}(v))^{-1}\ , \tag{19}$$

[10] We may always assume this. If the same physical task occurs in two jobs, we have to represent it by two distinct letters.

when the matrix $\mathcal{M}(v)$ is irreducible[11]*, and where $\rho(\mathcal{M}(v))$ is its (max,+) eigenvalue.*

The $|v|-1$ products of matrices in $\mathcal{M}(v)$ can be computed with a time complexity of $O(|v|(|\mathcal{J}|+|\mathcal{R}|))$, with sparse representations[12]. The standard way to compute the (max,+) eigenvalue $\rho(\mathcal{M}(v))$ is to use Karp algorithm [2, Th. 2.19], which runs in time $O(|\mathcal{J}|+|\mathcal{R}|)^3$. Thus, Theorem V.2 yields an algorithm with execution time $O(|v|(|\mathcal{J}|+|\mathcal{R}|)+(|\mathcal{J}|+|\mathcal{R}|)^3)$, which is *linear* in the length of the pattern $|v|$.

By comparison, the traditional method of [2], sketched in §V-A.1, consists first to build a special schedule dependent timed event graph, then to write the linear dater representation $x(n) = A_0 x(n) \oplus A_1 x(n-1)$, to eliminate the implicit part by $x(n) = \overline{A} x(n-1)$, with[13] $\overline{A} = A_0^* A_1$, and last to compute $\rho(\overline{A})$. Analyzing the execution time of this method[14], we found a total execution time of $O(|v|^2(|\mathcal{J}|+|\mathcal{R}|)+(|\mathcal{J}|+|\mathcal{R}|)^3)$, which is *quadratic* in the length of the pattern.

Note that we did not take into account the "modeling" complexities for both methods. However, the modeling aspect can be considered as a strong argument in favor of the automata-based algorithm. A new timed Event graph must be built for each new schedule in the traditional method, whereas the heap realization $\mathcal{M}$ is built only once, and remains valid for all (even non-periodic!) schedules.

Example V.3. Let us illustrate Theorem V.2 with the jobshop described in [18], § III. There are three machines: $\mathcal{R} = \{M_1, M_2, M_3\}$, four production sequences $\mathcal{J} = \{J_1, \dots, J_4\}$, $J_1 = a_1 a_2 a_3$, $J_2 = b_1 b_2$, $J_3 = c_1 c_2$, $J_4 = d_1 d_2$. The resource allocation and time execution maps are given in the following table:

Task	a_1	a_2	a_3	b_1	b_2	c_1	c_2	d_1	d_2
R	M_1	M_2	M_3	M_3	M_2	M_1	M_3	M_1	M_3
τ	1	3	3	1	2	2	1	2	1 .

One requires a production mix of $1/4, 1/4, 1/4, 1/4$ between J_1, J_2, J_3, J_4. A schedule satisfying this constraint (and compatible with the order of precedence of the elementary tasks) is v^ω, with $v = a_1 a_2 a_3 b_1 b_2 c_1 c_2 d_1 d_2$. The matrix of the resource automaton of a_1 is (omitting the $\mathbb{0}$ entries):

$$\mathcal{M}(a_1) = \begin{array}{c} \\ M_1 \\ M_2 \\ M_3 \\ J_1 \\ J_2 \\ J_3 \\ J_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} M_1 & M_2 & M_3 & J_1 & J_2 & J_3 & J_4 \end{array} \\ \begin{pmatrix} 1 & & & 1 & & & \\ & \mathbb{1} & & & & & \\ & & \mathbb{1} & & & & \\ 1 & & & 1 & & & \\ & & & & \mathbb{1} & & \\ & & & & & \mathbb{1} & \\ & & & & & & \mathbb{1} \end{pmatrix} \end{array}$$

[11] The extension to the reducible case is straightforward. The irreducibility assumption is always satisfied, unless there is a proper subset of jobs which does not share resources with other jobs.

[12] The matrices $\mathcal{M}(a)$, for $a \in \mathcal{T}$ differ from the identity matrix only on two row and two column indices. Thus, any matrix product $M\mathcal{M}(a)$ for a matrix M of size $|\mathcal{J}|+|\mathcal{R}|$ has a complexity of $O(|\mathcal{J}|+|\mathcal{R}|)$.

[13] Recall that A^* denotes the matrix $A^0 \oplus A^1 \oplus A^2 \oplus \dots$.

[14] We compute the columns of $A_0^* A_1$ using standard path algorithms (e.g. Jacobi algorithm, [17, Chap. 3, § 4.1]). Note that one need not compute $A_0^* A_1$ but only a $C \times C$ sub-block of $A_0^* A_1$, where C is the set of transitions with at least one token in one downstream place. Hence, we only have to compute $|C| \leq |\mathcal{J}|+|\mathcal{R}|$ columns of $A_0^* A_1$, which takes time $O((|\mathcal{J}|+|\mathcal{R}|)|v|^2)$.

We leave it to the reader to write the other matrices, and just give

$$\mathcal{M}(v) = \begin{pmatrix} 7 & 10 & 9 & 5 & 10 & 10 & 10 \\ \mathbb{0} & 3 & 2 & \mathbb{0} & 3 & 3 & 3 \\ \mathbb{0} & \mathbb{0} & 3 & 4 & \mathbb{0} & 5 & 5 \\ 7 & 10 & 9 & 5 & 10 & 10 & 10 \\ 6 & 9 & 8 & \mathbb{0} & 9 & 9 & 9 \\ 3 & 6 & 5 & \mathbb{0} & 6 & 6 & 6 \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & 2 & \mathbb{0} & 3 & 3 \end{pmatrix} .$$

We have $\rho(\mathcal{M}(v)) = 9$, which yields the throughput of $1/9$, as in §V of [18]. Enumerating all the admissible patterns (composed of one production sequence of each type), we find an optimal throughput of $1/7$, attained for instance for the cyclic schedule $\tilde{v}^\omega$, with $\tilde{v} = a_1 d_1 b_1 a_2 b_2 c_1 a_3 d_2 c_2$.

Remark V.4. It is essential to note that the heap representation coincides (up to a $90°$ rotation) with a version of the Gantt charts, where *both* the occupation of jobs and machines are represented. Traditional Gantt charts are exactly the restriction to the machine columns of heaps of pieces representations.

As an illustration, we have represented below the traditional Gantt chart for the model treated at length in this paper (Ex. II.2, Ex. II.6, §V-A) and the schedule $abcdcdabab$.

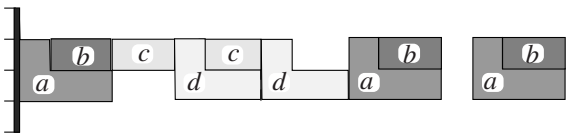


Fig. 10. (Conventional) Gantt charts are not (max,+) linear.

This representation of dimension two does not correspond to a (max,+) automata. The smallest heap representation is of size three, see Fig. 5,(3).

Concluding remarks

Let us indicate some possible developments of this work. 1. It remains to develop heuristics and performance bounds for scheduling, based on heap & automata representations. 2. Matrix representations allow us to apply standard time parallelization methods (breaking long matrix products in subproducts, mapped on independent processors) to the parallel simulation of stochastic Petri nets. 3. Automata results (e.g. worst case performance evaluation, as in [15]) can be immediately applied to heap models. 4. The main limitation of the automata-modeling presented here is the restriction to safe Petri nets. A natural question consists in characterizing the subclasses of non safe Petri nets for which such an automata modeling remains valid.

Acknowledgments

This work was initiated at BRIMS, Hewlett-Packard Laboratories, Bristol, were the first author was visiting and the second author was doing a post-doc. The authors thank Jeremy Gunawardena for his hospitality during their sojourn at BRIMS. The authors thank the referees for their constructive comments.

References

[1] M. Ajmone-Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Trans. on Software Engin.*, 15(7):832–846, 1989.

[2] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity*. John Wiley & Sons, New York, 1992.

[3] F. Baccelli, S. Foss, and B. Gaujal. Structural, temporal and stochastic properties of unbounded free-choice Petri nets. Technical Report 2411, INRIA, Sophia Antipolis, France, 1994. To appear in IEEE TAC.

[4] J. Berstel and C. Reutenauer. *Rational Series and their Languages*. Springer Verlag, 1988.

[5] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models and Their Properties*, volume 254 of *Lect. Notes in Computer Sciences*. Springer-Verlag, 1987.

[6] M. Brilman and J.M. Vincent. Synchronisation by resources sharing : a performance analysis. Technical report, MAI-IMAG, Grenoble, France, 1995.

[7] J. Carlier and P Chretienne. Timed Petri net schedules. In *Advances in Petri Nets*, number 340 in LNCS, pages 62–84. Springer Verlag, 1988.

[8] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. Analyse du comportement périodique des systèmes de production par la théorie des dioïdes. Technical Report 191, INRIA, 1983.

[9] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. A linear system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. *IEEE Trans. Automatic Control*, AC-30:210–220, 1985.

[10] G. Cohen, S. Gaubert, and J.P. Quadrat. Algebraic system analysis of timed Petri nets. In J. Gunawardena, editor, *Idempotency*. Cambridge University Press, 1997.

[11] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Comp. Sc.* Cambridge Univ. Press, 1995.

[12] V. Diekert. *Combinatorics on traces*. Number 454 in LNCS. Springer Verlag, 1990.

[13] V. Diekert and G. Rosenberg, editors. *The book of traces*. World Scientific Publ., 1995.

[14] S. Eilenberg. *Automata, languages and machines*, volume A. Academic Press, New York, 1974.

[15] S. Gaubert. Performance evaluation of (max,+) automata. *IEEE Trans. on Automatic Control*, 40(12), Dec 1995.

[16] S. Gaubert and J. Mairesse. Task resource models and (max,+) automata. In J. Gunawardena, editor, *Idempotency*. Cambridge University Press, 1997.

[17] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, Paris, 1979. Engl. transl. *Graphs and Algorithms*, Wiley, 1984.

[18] H. Hillion and J.M. Proth. Performance evaluation of job shop systems using timed event graphs. *IEEE Trans. Automatic Control*, 34(1):3–9, 1989.

[19] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, University of Washington, 1995.

[20] A. Mazurkiewicz. Concurrent program schemes and their interpretations. Research report DAIMI Rep. PB-78, Aarhus Univ., 1977.

[21] A. Mazurkiewicz. Trace theory. In *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in LNCS, pages 279–324. Springer Verlag, 1987.

[22] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[23] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[24] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, MIT, Boston, 1974.

[25] A. Salomaa and M. Soittola. *Automata Theoretic Aspects of Formal Powers Series*. Springer Verlag, Berlin, 1978.

[26] G.X. Viennot. Heaps of pieces, I: Basic definitions and combinatorial lemmas. In Labelle and Leroux, editors, *Combinatoire Énumérative*, number 1234 in Lect. Notes in Math., pages 321–350. Springer, 1986.

[27] J.M. Vincent. Some ergodic results on stochastic iterative DEDS. Technical Report 4, Apache, IMAG, Grenoble, 1994. To appear in JDEDS.