An approach based on timed Petri nets and tree encoding to implement search algorithms for a class of scheduling problems

Dimitri Lefebvre, Francesco Basile

Version of Record: https://www.sciencedirect.com/science/article/pii/S0020025521000190
Manuscript_9291e9af8bb0776bcd6924fa16ad2f01

An approach based on Timed Petri nets and tree encoding to implement search algorithms for a class of scheduling problems

Dimitri Lefebvre[*], Francesco Basile[**]

[*]*Normandie Univ, UNIHAVRE, GREAH, 76600 Le Havre, France (dimitri.lefebvre@univ-lehavre.fr)*

[**]*DIEM, Università di Salerno, Italy (fbasile@unisa.it)*

Abstract: Scheduling problems have been approached several times by Petri nets. Indeed, the usage of a Petri net model guarantees the feasibility of the candidate solutions, but it does not provide an accurate evaluation of the time required to complete the considered workshop. For this purpose, a class of timed Petri nets, namely *structured nets*, is defined and the encoding of the structure and time information of such nets as a tree is presented. This encoding, in combination with the resolution of some linear matrix inequalities, is used to estimate the residual time required to complete each job of the considered workshop. The main advantage of this computation is to provide an estimation of the residual time as an interval that includes necessarily the exact residual duration. Consequently, the lower bound of the interval never overestimates the exact duration and can be used as a part of the cost function involved in many exploration algorithms as the A* or the Beam Search algorithms. In this paper, the proposed estimation function is used with the Hybrid Filtered Beam Search algorithm and performances are discussed for several examples of workshops. The paper also illustrates that the approach can be combined with supervisory control to accelerate the convergence of the exploration since deadlocked solutions can be eliminated directly in the model.

Keywords: Flexible manufacturing systems, scheduling, Timed Petri nets, Beam search algorithm

1. INTRODUCTION

Scheduling is one of the most important and difficult issues in the operation of flexible manufacturing systems (FMSs) due to routing flexibility and shared resources. It belongs to the class of NP-hard combinatorial optimization problems. Petri Nets (PNs) have demonstrated their ability to handle such problems with numerous contributions. PNs have an underlying mathematical structure that can be exploited to perform qualitative and quantitative analysis of the systems. They can be directly converted into simulation models; in addition, they are graphical, easy to develop, extend, and offer a better understanding of the dynamic behavior of the systems by means of tokens and transitions. The major advantages of using PNs in the scheduling of production systems over other tools include the ability to represent many states in a concise manner, to capture precedence relations and structural interactions, and to model deadlocks, conflicts, buffer-sizes, and multi-resource constraints. Activities, resources, and constraints of a manufacturing system can be represented in a single coherent formulation. Concurrent and asynchronous activities, resources sharing, routing flexibility, and complex constraints on process sequences can be explicitly and concisely modeled by PNs.

As far as scheduling problems are considered, a large variety of PN subclasses have been listed and discussed [29], [30]. Job shop problems are the most popular scheduling problems and Systems of Simple Sequential Processes with Resources (S^3PR)

have been defined as a subclass of ordinary and conservative PNs [11] for this class of problems. Later, to extend the use of resources in job shops, a generalization of S^3PR models called Systems of Sequential Systems with Shared Resources (S^4R) has been defined [2]. Scheduling problem for totally flexible workshops have been also considered in particular for open shops that consist of several jobs which are processed on multiple resources with full routing flexibility. A subclass of PN models called Set of Simple Open Processes with Resources (S2OPR) has been defined for such problems [34]. In this paper a model suitable to describe workshops that have more flexibility than job shops but less flexibility than open shops is proposed. We refer to such jobs as *structured jobs*. We propose to model the structured jobs with a class of timed Petri nets [10] referred to as *structured nets* and to abstract interesting structural and timing properties of the net in a tree, namely a *structure tree*. This abstraction is different from the existing works that aim to reduce the PN structure [28], [35], [44], [46]. The aim of PN reduction is to develop models of smaller size but preserving the precise representation of the allocation of the resources within the operations. The structure tree is also different from net unfolding methods [6], [19]. Compared to net unfolding, the tree encoding takes into account the effects of events interleaving (as net unfolding already does) and filters it. But, in addition, at the same level, the structure tree incorporates timed information essential to estimate the duration of sequences of transitions and to generate the schedule.

Basically, the contributions of the paper are double: first, to detail a systematic modeling methodology, based on structure trees, for a certain class of flexible workshops with parallelisms and choices; second, to use the advantages of the structure trees combined with pruning search algorithms and supervisory control to generate admissible and efficient schedule at lowest cost. The precedence constraints and operating times of the considered workshops are both encoded in the structure trees. Our work prompts the research effort in the domain of scheduling: the timing properties and rapid evaluation of the structure trees make them tractable for use with search algorithms to compute, rapidly, good schedule candidates even if the search algorithm is an aggressive pruning one that eliminates most of the candidates.

Based on PN models, there are mainly two categories of methods devoted to scheduling problems with PNs: the first one uses the PN model to reveal and analyze properties in order to decide/ensure that a schedule can be efficiently found; the second one uses the PN model to design a search algorithm and obtain the schedule. The method proposed in this paper belongs to the latter.

In the first category, much effort has been done for scheduling cluster tools and one can refer to [37] for a recent review about the main contributions in this area. Most of the results have been obtained by analytical expressions if some schedulability conditions are satisfied [47], [48], [49]. Linear programming has been intensively used for that purpose. Revisiting processes [49] and non-revisiting ones [38], [39] have been studied with or without residency times constraints [50]. Activity time variations have been also taken into consideration [51].

Methods in the second category generally use the PN reachability graph. However, due to exponential growth of the number of states, generating the entire reachability graph is time consuming even for small size systems. Lee and DiCesare in [20] pioneered the domain by proposing to combine the PN reachability graph and the A^* algorithm to schedule FMSs. However, due to the problem of state explosion, this method was not tractable for large systems. To find schedules, researchers try to reduce the state space by using search algorithms that include heuristic functions, and generate only a portion of the reachability graph. These methods can be classified as "*PNs with a search algorithm*" [43] and the contribution of this paper can be framed in this context.

Some improvements concern the sizing and delimitation of the parts in reachability graph to explore and other ones concern the definition of the heuristic function used to sort and select the best candidates. Improvements in the first category have been obtained in several ways as for example, by constraining the number of unexplored markings in a fixed range [42]; by combining the A* search with backtracking strategy [52]; by restraining the search in a limited local search window [41]; by pruning the non-promising branches [25], [31]; by unfolding the PN model [19], [45]. Contrary to the A* algorithm where all explored candidates are conserved, an informed graph search referred to as Beam Search (BS) algorithm [36], conserves only a selection of the best previously explored candidates. The complexity in time of BS algorithm remains polynomial thanks to this restriction. But the main drawback is that good candidates could be eliminated due to the selection strategy. Several variants of the BS algorithm have been proposed in combination with PNs [32]. Improvements of the heuristic function have been also studied, for example with a function based on PN state equations [17], or based on artificial intelligence [55] and genetic algorithms [53]. Other techniques based on non-admissible functions [16], have been also developed. A simple and efficient heuristic function based on the PN paths has been proposed for job shop problems [31]. The performance of the previous methods depends strongly on the choice of the heuristic function used as a metric to evaluate the distance from the current state to the reference state because this function is used to prune branches and limit the exploration. In this work, an efficient variant of beam search methods – namely Hybrid Filtered Beam Search (HFBS) – [33] is used and a new heuristic function based on the tree encoding of the structured nets is detailed.

A critical issue affecting the solution of a scheduling problem is the existence of deadlocks since they may prevent the expected schedule to reach the objective. In particular, deadlocks occur frequently in workshops without intermediate buffer between operations (in that case the reservation and release of resources is simultaneous and additional constraints occur). In order to prevent deadlocks (or to restrict the possible behaviors of the system in a certain sense), monitor places are usually added to the TPN model. Such control places act as a supervisor that constrains the system to remain in some admissible regions [3], [12], [18], [26]. Note that supervisory control can be used to enforce other types of specifications on a PN model, such as the capacity of shared resources, the maximum number of jobs simultaneously allowed, liveness, controllability, etc. These specifications collectively define a set of legal reachable markings and a set of forbidden markings to be avoided.

A popular approach is to define the marking specification with a set of Generalized Mutual Exclusion Constraints (GMECs) that restricts the set of legal markings to a convex region of the reachability graph. A PN supervisor can be calculated that enforces the previous constraints using monitor places [4], [12], [54]. A more general form of constraint specification is a disjunction of GMECs [5], [9]. Supervisors by means of logical predicate, duplicated transitions [13], or inhibitor arcs [8] can be also defined. In general, a PN supervisor may be not maximally permissive, and so it eliminates not only the forbidden markings but also some markings that could belong to the optimal schedule. Consequently, the scheduling of the controlled net may be degraded and deadlock free scheduling remains an important topic to study. The work in [1] is one of the first that uses timed PNs to model FMSs and proposes a deadlock-free scheduling algorithm. The algorithm is based on the depth-first search strategy together with a siphon truncation technique. Then, genetic algorithms based on PNs [15] and a model predictive approach [21] have been also studied to compute efficient schedules that avoid the deadlocks. Deadlock free scheduling remains out of the scope of this paper but we explain how to combine the proposed exploration method with supervisory control in order to eliminate a priori any possible deadlock and other forbidden markings. The paper illustrates that using a supervisory control generally accelerates the convergence but may degrade the makespan (i.e., the duration to perform all operations of the jobs).

This paper extends our preliminary study about the PN structure encoding with trees [22]. Compared to [22], the new results presented here are first to provide a formal definition of a subclass of nets, namely structured PNs. Then, based on structured

nets, we propose a systematic modeling methodology, based on trees, for some scheduling problems in which each job may include some choices and parallelisms between the operations. In particular, we prove that the tree that encodes the model is unique. Taken advantage from the temporal properties of the tree, we use it with the HFBS algorithm to form a new algorithm that searches for optimal schedules. Finally, we show how to combine the new search algorithm with supervisory control in order to increase the rapidity of the search. In the whole, these new results lead to a systematic scheduling approach that is suitable for a certain class of problems and that remains tractable for medium-size to large-size systems.

The rest of the paper is organized as follows. In Section 2, the scheduling problems with choices and parallelisms that are considered in this paper are described. In Section 3, preliminaries about timed Petri net are presented. Structured nets are formally introduced. Section 4 details the modelling of the considered scheduling problems with structured nets and structure trees. Section 5 is devoted to the use ofE the tree encoding to estimate the residual duration to complete a given job with choices and parallelisms between operations. In Section 6, this estimation is used together with the HFBS method in order to solve scheduling problems for FMSs composed of structured jobs. In Section 7, several examples are considered to illustrate the performance of the approach. A comparison with the usual path-based heuristic function is discussed and the approach is also combined with supervisory control.

2. STRUCTURED FMSs

In this section we characterize the FMSs we are interested in. An FMS is basically a workshop composed of a set of jobs that executes some operations according to the availability of some resources. Let $\boldsymbol{J}$, $\boldsymbol{O}$ and $\boldsymbol{R}$ be respectively the set of jobs, the set of operations and the set of resources. Each job J_k consists in the execution of a set of operations $o_i \in \boldsymbol{O}$ in a certain order and each operation needs some resources $r_j \in \boldsymbol{R}$ to be performed. Formally an operation o_i is characterized by $o_i = (d_i, R_i)$, the duration d_i represents the minimal time needed to execute o_i (once the required resources are available) and $R_i \subseteq \boldsymbol{R}$ is the subset of resources that are required to perform the operation o_i. A given resource r_j may be required by several operations at the same time. In that case, one has to decide to which operation the resource must be allocated first. Buffers may also exist between successive operations.

Here, we consider one or several executions of a given job (in both a simultaneous or serial mode) with total precedence constraints between its operations and represent such a job by (*O, cap, exe*) where *O* is a set of operations (each one defined with a duration and a set of resources). The operations in *O* need to be processed in a specific order (also known as the precedence constraints of the job) induced by the indexes of the operations (i.e., o_i follows o_{i-1} and precedes o_{i+1}). The parameter *cap* is the lot size of the job (i.e. the maximal number of possible simultaneous executions of the set of the operations), *exe* is the number of times the job has to be performed according to the scheduling objective.

In this paper we are interested in a less restrictive class of scheduling problems where the jobs include choices and also parallelisms between sequences of operations. The combination of these structures leads to jobs that are characterized by partial precedence constraints and partial routing flexibility.

A particular class of jobs, namely *structured jobs*, is formally introduced next to describe such workshops. For this purpose, three basic *composition functions* are introduced.

Definition 1: Let o_1, o_2 be a pair of operations in a given job.

- The *serial basic function* $S(o_1, o_2)$ returns a job J as the sequence of o_1 and o_2. The two operations are sequentially performed according to the order induced by the indexes of the operations (i.e., o_1 is performed first, then o_2).
- The *choice basic function* $C(o_1, o_2)$ returns a job J as the choice between the operations o_1 and o_2: only one operation is performed within $\{o_1, o_2\}$.
- The *parallel basic function* $P(o_1, o_2)$ returns a job J as the parallel execution of o_1 and o_2: the two operations are concurrently performed and the order is not specified.

Note that the basic composition functions can be trivially extended so that they are applicable not only to operations but also to sets of operations that have been already structured according to the same compositions functions. The *structure function* $F(O)$ is defined recursively as a tree from the recursive application of the preceding composition functions.

Definition 2: Let $O = \{o_1, o_2,\ldots,o_k\}$ be the set of k operations in a given job and S,C,P being the three composition functions introduced with Definition 2. A *structure function* is a set F such that each element $j \in F$ is either:

- An operation : $j \in O$
- A 4-tuple j= (*label, type, first, second*) where *label* is the label of element j; *type* $\in \{S,C,P\}$ is the type of the composition function used to define the element j; *first* and *second* are two structure functions called respectively first and second structure functions.

If the element j is of type S, then j results from *first* then *second*. If j is of type C, then j results from *first* or *second*. Finally, if j is of type P, then j results from *first* and *second*. *Structured jobs* are defined consequently.

Definition 3: A *structured job J* is defined as (*O, cap, exe, F*) where O is a subset of operations, *cap* is the lot size of the job, *exe* is the number of times the job has to be performed according to the scheduling objective and F is a structure function that specifies the routing flexibility between the operations and is defined over the set of operations O and according to the composition functions *S, C,* and *P.*

Note that the term "*structured*" job focuses on the fact that the operations respect a partial order. This order specifies for each operation when it can be performed with respect to the other operations of the job. The definition of a structured FMS results straightforward from Definition 3:

Definition 4: A *structured FMS* is defined as (***J, R***) where ***J*** is a set of structured jobs and ***R*** is a set of resources shared by the jobs in ***J***.

The considered class of structured FMSs is wide enough to include many situations of practical interest including not only transformation processes but also assembly or disassembly workshops. Structured FMSs describe workshops that have more flexibility than job shop problems but less flexibility than open shop problems.

Example 1: Consider as an example, a job with 7 operations $\boldsymbol{O} = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$. The precedence constraints can be described as follows: perform o_1 before o_2; perform o_3 before o_4. Then, some more complicated constraints are introduced:

perform operation o_5 simultaneously with the set of operations $\{o_1, o_2, o_3, o_4\}$; perform either the set of operations $\{o_1, o_2\}$ or $\{o_3, o_4\}$. Finally, perform o_7 before the set of operations $\{o_1, o_2, o_3, o_4, o_5\}$ and $\{o_1, o_2, o_3, o_4, o_5\}$ before o_6.

The resulting structure function F of Example 1 is detailed in Fig. 1. Observe that all precedence constraints are encoded in this function according to the three composition functions S, C and P and according to the arbitrary order used to enumerate the operations of Example 1.

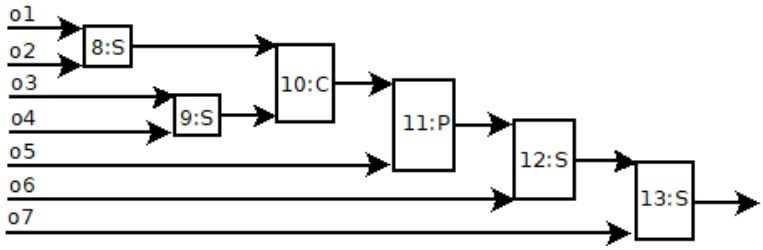

Fig. 1. Structure function F for the job of Example 1

3. STRUCTURED TPNs

In this section we introduce a new class of timed Petri nets – namely *structured TPNs* – that will be used to model structured FMSs. This modeling takes its inspiration from the S^3PR and S^4R models.

3.1 Timed Petri nets

First, we recall some basic definitions about PNs and TPNs.

Definition 6 : A *PN* is defined as $G = <\boldsymbol{P}, \boldsymbol{T}, W_{PR}, W_{PO}>$, where $\boldsymbol{P} = \{p_1,\ldots, p_n\}$ is a set of n places and $\boldsymbol{T} = \{t_1,\ldots, t_q\}$ is a set of q transitions with indexes $\{1,\ldots q\}$, $W_{PO} \in (\mathbf{N})^{n\times q}$ and $W_{PR} \in (\mathbf{N})^{n\times q}$ are the post and pre incidence matrices ($\mathbf{N}$ is the set of non-negative integer numbers), and $W = W_{PO} - W_{PR}$ is the incidence matrix. $<G, M_I>$ is a *PN system* with initial marking M_I.

The vector $M \in (\mathbf{N})^n$ represents the PN marking. In addition, $P(M)$ stands for the support of marking M (i.e. the subset of places that have a non-zero number of tokens at M). The enabling degree of transition t_j at marking M is given by $n_j(M) = \min\{\lfloor m_k / w^{PR}_{kj} \rfloor : p_k \in {}^\circ t_j\}$ where ${}^\circ t_j$ stands for the preset of t_j (i.e., set of input places), m_k is the marking of place p_k, w^{PR}_{kj} is the entry of matrix W_{PR} in row k and column j. A transition t_j is enabled at marking M if $n_j(M) \geq 1$, this is denoted as $M[t_j>$. When t_j is enabled it may fire, this is denoted by $M[t_j>M'$ and M' can be computed as $M' = M + W.X(t_j)$ where $X(t_j) \in (\mathbf{N})^q$ is the firing count vector corresponding to the firing of t_j. A firing sequence σ fired at marking M_I is defined as $\sigma = t(j_1)\, t(j_2)\ldots t(j_h)$ where $j_1,\ldots j_h$ are the indexes of the transitions. $X(\sigma)$ is the firing count vector associated to σ that specifies the number of times each transition appears in σ, $|\sigma| = h$ is the length of σ, and $\sigma = \varepsilon$ stands for the empty sequence. A marking M is said reachable from initial marking M_I if there exists a firing sequence σ such that $M_I[\sigma>M$ and σ is said feasible at M_I. $\boldsymbol{R}(G, M_I)$ is the set of all reachable markings from M_I.

Timed Petri nets are PNs whose behaviors are constrained by a timing structure [10]. For this reason, timed PNs have been intensively used to describe discrete event systems like production systems [7]. This paper concerns T-timed PNs (TPNs) [40].

Definition 7 : A TPN system is defined as $<G, M_I, D>$ where $<G, M_I>$ is a PN system and $D \in (\mathbf{R}^+)^q$ is the vector of the firing delays of the transitions ($\mathbf{R}^+$ is the set of non-negative real numbers): the jth entry of D –namely d_j – represents the minimum firing delay of transition t_j. If t_j is enabled at a given time τ, it does not fire before $\tau + d_j$; if it fires at a time instant $\tau' \geq \tau$, we call *remaining duration* the duration $\delta = \max(0, \tau + d_j - \tau')$.

The considered TPNs have a time semantic defined according to an infinite server policy and an enabling memory policy [14]. In case of choice between two transitions, a preselection is used to decide which one fires. Finally, the considered TPNs behave with an *earliest firing policy*: consider a fireable sequence $\sigma = t(j_1)\ t(j_2)\ldots t(j_h)$, then each transition $t(j_k)$ of the sequence fires at earliest when (1) the preceding transition $t(j_{k-1})$ in σ has fired; (2) the remaining duration of $t(j_k)$ is 0.

Definition 8: A *path* $pth = x_1\ x_2\ \ldots\ x_K$ in a PN is an orderly sequence of K nodes with $x_k \in \boldsymbol{T} \cup \boldsymbol{P}$ and $x_{k+1} \in (x_k)^\circ$ for $k = 1,\ldots,K-1$.

The duration of a path *pth* is defined by:

$$\chi(pth) = \sum_{t_j \in (pth \cap \boldsymbol{T})} d_j \qquad (1)$$

The set of paths from departure node x_1 to destination node x_K is referred to as $PTH(x_1, x_K)$. A path $pth \in PTH(x_1, x_K)$ is said of minimal duration if there does not exist any other path in $PTH(x_1, x_K)$ with a smallest duration. We refer to a path of minimal duration from x_1 to x_K as to $pth^*(x_1, x_K)$ and to its duration as $\chi^*(x_1, x_K)$.

Example 2: Fig. 2-left is an example of TPN with a set of places $\boldsymbol{P} = \{p_1, \ldots, p_9\}$ and a set of transitions $\boldsymbol{T} = \{t_1, \ldots, t_7\}$. Matrices W_{PR}, W_{PO} are both of dimensions 9×7. For example, $W_{PR}(p_7, t_7) = W_{PR}(p_8, t_7) = W_{PO}(p_1, t_7) = W_{PO}(p_3, t_7) = 1$ describes how transition t_7 is connected to the rest of the net. The vector of the minimal firing delays of the transitions is defined as $D = (2\ 4\ 0\ 7\ 9\ 8\ 7)^T$ and the time delays are shown next to the transitions in Fig. 2-left so that "t_1: 2" means that transition t_1 should be enabled during 2 Time Units (TU) before it fires. The initial marking M_I in this example is such that $M(p_7) = 2$ and $M(p_8) = 1$. The transition t_7 is enabled at M_I with a degree 1. An example of path in $PTH(p_1, p_6)$ is $pth = p_7\ t_7\ p_3\ t_1\ p_4\ t_2\ p_6$ with duration $\chi(pth) = d_7 + d_1 + d_2 = 13$ TU. Note that another path exists in $PTH(p_1, p_6)$: $pth' = p_7 t_7 p_3\ t_3\ p_5\ t_4\ p_6$ of duration $\chi(pth') = d_7 + d_3 + d_4 = 14$ TU. An example of valid firing sequence is $\sigma = t_7\ t_1\ t_5$ that will lead the system from M_I to marking M such that $M(p_7) = M(p_4) = M(p_2) = 1$. When this sequence fires with earliest firing policy, its duration is 16 TU.

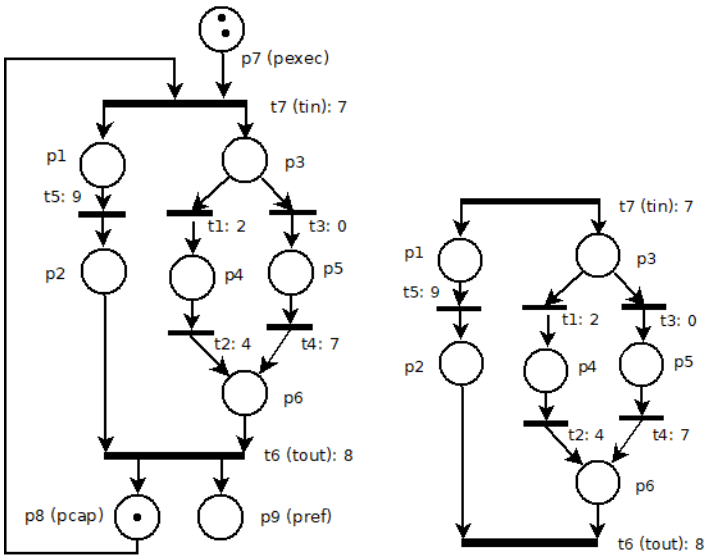

Fig. 2. An example of extended structured TPN (left); an example of structured TPN (right)

3.2 Structured TPNs

In this section we introduce a subclass of TPN models that will be used to model the structured jobs. Let us first introduce the following elementary PN structures that will be used to represent the composition functions *S, C* and *P* introduced in the previous section.

Definition 9: Let $G = \langle \boldsymbol{P}, \boldsymbol{T}, W_{PR}, W_{PO} \rangle$ being a PN.

- An *elementary sequence* (Fig. 3a) is a pair of transitions $t_{j1}, t_{j2} \in \boldsymbol{T}$ such that there exists a place $p_i \in \boldsymbol{P}$ with: (1) $t_{j1}^{\circ} = {}^{\circ}t_{j2} = \{p_i\}$, (2) ${}^{\circ}p_i = \{ t_{j1}\}$, (3) $p_i^{\circ} = \{ t_{j2}\}$.
- An *elementary choice* (Fig. 3b) is a pair of transitions $t_{j1}, t_{j2} \in \boldsymbol{T}$ such that there exist 2 places $p_{i1}, p_{i2} \in \boldsymbol{P}$ with: (1) ${}^{\circ}t_{j1} = {}^{\circ}t_{j2} = \{p_{i1}\}$; (2) $t_{j1}^{\circ} = t_{j2}^{\circ} = \{p_{i2}\}$.
- An *elementary parallelism* (Fig. 3c) is a pair of transitions $t_{j1}, t_{j2} \in \boldsymbol{T}$ such that there exist 4 places $p_{i1}, p_{i2}, p_{i3}, p_{i4} \in \boldsymbol{P}$ with: (1) ${}^{\circ}t_{j1} = \{p_{i1}\}$ and $t_{j1}^{\circ} = \{p_{i2}\}$ and ${}^{\circ}t_{j2} = \{p_{i3}\}$ and $t_{j2}^{\circ} = \{p_{i4}\}$, (2) ${}^{\circ}p_{i1} = \{t_{j3}\}$ and $p_{i1}^{\circ} = \{t_{j1}\}$ and ${}^{\circ}p_{i3} = \{t_{j3}\}$ and $p_{i3}^{\circ} = \{t_{j2}\}$ and ${}^{\circ}p_{i2} = \{t_{j1}\}$ and $p_{i2}^{\circ} = \{t_{j4}\}$ and ${}^{\circ}p_{i4} = \{t_{j2}\}$ and $p_{i4}^{\circ} = \{t_{j4}\}$.

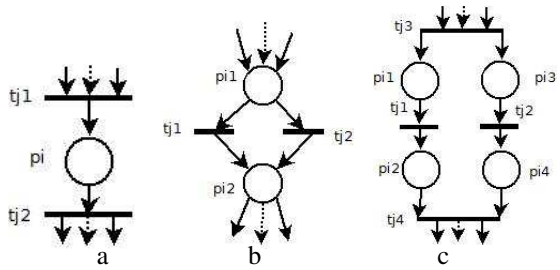

Fig. 3: Elementary PN structures: sequence (a), choice (b), parallelism (c)

For any $G = \langle \boldsymbol{P}, \boldsymbol{T}, W_{PR}, W_{PO} \rangle$ and subset $\boldsymbol{T'} \subseteq \boldsymbol{T}$ with q' transitions, let us define $W_{PR}(\boldsymbol{P},\boldsymbol{T'}) \in (\mathbf{N})^{n \times q'}$ and $W_{PO}(\boldsymbol{P},\boldsymbol{T'}) \in (\mathbf{N})^{n \times q'}$ as the submatrices of W_{PR} and W_{PO} that are restricted to the transitions in $\boldsymbol{T'}$. A *nested structure* is recursively defined as:

Definition 10: ($\boldsymbol{T'}$, $W_{PR}(\boldsymbol{P},\boldsymbol{T'})$, $W_{PO}(\boldsymbol{P},\boldsymbol{T'})$) is a *nested structure* if a partition of $\boldsymbol{T'} = \boldsymbol{T'_1} \cup \boldsymbol{T'_2}$ exists such that:

- $|\boldsymbol{T'_1}| = 1$ or ($\boldsymbol{T'_1}$, $W_{PR}(\boldsymbol{P}, \boldsymbol{T'_1})$, $W_{PO}(\boldsymbol{P}, \boldsymbol{T'_1})$) is a nested structure,
- $|\boldsymbol{T'_2}| = 1$ or ($\boldsymbol{T'_2}$, $W_{PR}(\boldsymbol{P}, \boldsymbol{T'_2})$, $W_{PO}(\boldsymbol{P}, \boldsymbol{T'_2})$) is a nested structure,
- If $|\boldsymbol{T'_1}| = 1$ and $|\boldsymbol{T'_2}| = 1$, then $\{\boldsymbol{T'_1}, \boldsymbol{T'_2}\}$ is either an elementary sequence, an elementary choice, or an elementary parallelism.

Definition 11: Let $G = <\boldsymbol{P}, \boldsymbol{T}, W_{PR}, W_{PO}>$ being a PN, G is a *structured* PN if:

- G is an ordinary PN (i.e. arc weights equal 1),
- G is an acyclic net with a single source transition, namely t_{in}, and a single sink transition, namely t_{out},
- ($\boldsymbol{T}$, W_{PR}, W_{PO}) is a nested structure.

Observe that structured PNs are different from workflow nets [44], the latter do not require to be structured.

4. MODEL OF STRUCTURED FMSs WITH TPNs

In this section, structured FMSs without intermediate buffer are modeled with TPNs. For this purpose, we represent each *structured job* of an FMS with a *structured* net. Note that there is no difficulty to extend this modeling schema to structured FMSs with intermediate buffers (in that case the reservation and release of the resources is separated and deadlocks do no longer occur in the possible schedules).

4.1 Model of a structured job

An operation o_j is modeled by a transition t_j as shown in Fig.6. The operation o_j has an input buffer represented by a place $p_{in\,j}$ and an output buffer represented by a place $p_{out\,j}$ (that is in the same time the input buffer of the next operation of the job when intermediate buffers do not exist). The duration d_j of o_j is represented by the firing delay $D(t_j)$ of t_j. In addition, each resource r_k used to perform o_j is represented by an additional place p_{rk} that is simultaneously in the preset and postset of t_j (and can also belong to the preset and postset of other transitions to model resources that are shared by several operations). Consequently, a set $\boldsymbol{O}$ of q operations is represented by a set $\boldsymbol{T}$ of q transitions.

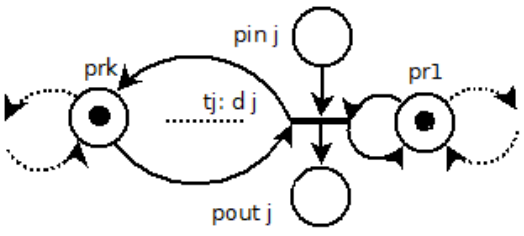

Fig.6: Operation modeling with TPNs

The modeling process of a structured job ($\boldsymbol{O}$, *cap*, *exe*, F) with a structured net is based on the use of the structure function F. The construction starts with the set of transitions $\boldsymbol{T}$ that results from the modeling of the operations and from an empty set of places $\boldsymbol{P}$ and the empty matrices W_{PR} and W_{PO}. Each node (j, *type*, j_1, j_2) of the structure function with *type* $\in \{S, C, P\}$ is successively used to create new places and define iteratively the incidence matrices of the TPN model.

- If *type* =*S*, then a place p_i is added in $\boldsymbol{P}$ with $^\circ p_i = \{ j_1\}$ and $p_i^\circ = \{ j_2\}$. A row vector i of size q is added to the matrices W_{PR} and W_{PO} and $W_{PO}(i, j_1) = W_{PR}(i, j_2) = 1$ (the others entries of row i being 0).
- If *type* =*C*, then 2 places p_{i1}, p_{i2} are added in $\boldsymbol{P}$ with $p_{i1}^\circ = p_{i2}^\circ = \{ j_1, j_2 \}$. Two row vector i_1 and i_2 of size q are added to the matrices W_{PR} and W_{PO} and $W_{PO}(i_1, j_1) = W_{PO}(i_1, j_2) = W_{PR}(i_2, j_1) = W_{PR}(i_2, j_2) = 1$ (the others entries of rows i_1 and i_2 being 0).
- If *type* =*P*, then 4 places p_{i1}, p_{i2}, p_{i3}, p_{i4} are added in $\boldsymbol{P}$ with $^\circ p_{i1} = \{t_{j3}\}$, $p_{i1}^\circ = \{t_{j1}\}$, $^\circ p_{i3} = \{t_{j3}\}$, $p_{i3}^\circ = \{t_{j2}\}$, $^\circ p_{i2} = \{t_{j1}\}$, $p_{i2}^\circ = \{t_{j4}\}$, $^\circ p_{i4} = \{t_{j2}\}$ and $p_{i4}^\circ = \{t_{j4}\}$. Four row vector i_1 to i_4 of size q are added to the matrices W_{PR} and W_{PO} and $W_{PO}(i_1, j_1) = W_{PO}(i_3, j_2) = W_{PR}(i_2, j_1) = W_{PR}(i_4, j_2) = 1$ (the others entries of rows i_1 to i_4 being 0).

In order to end the model of a structured job, the acyclic structured net is completed with three particular places: p_{exec} whose initial marking represents the number n of times the job is expected to be performed, p_{cap} whose initial marking represents the lot size of the job, and p_{ref} that counts the total number of executions of the job. The resulting TPN model is defined as an *extended structured net*:

Definition 12: Let $G = \langle \boldsymbol{P}, \boldsymbol{T}, W_{PR}, W_{PO} \rangle$ being a PN and $\boldsymbol{P} = \boldsymbol{P'} \cup \{ p_{exec}, p_{cap}, p_{ref} \}$ being a partition of the set of places. G is an *extended structured PN* if:

- $G' = \langle \boldsymbol{P'}, \boldsymbol{T}, W'_{PR}, W'_{PO} \rangle$ is a structured net, where $W'_{PR} = W_{PR}(\boldsymbol{P'}, \boldsymbol{T})$ and $W'_{PO} = W_{PO}(\boldsymbol{P'}, \boldsymbol{T})$ (i.e. the submatrices extracted from W_{PR} and W_{PO} when only the places in $\boldsymbol{P'}$ are considered),
- p_{exec} is such that $^\circ(p_{exec}) = \emptyset$ and $(p_{exec})^\circ = t_{in}$,
- p_{cap} is such that $^\circ(p_{cap}) = t_{out}$ and $(p_{cap})^\circ = t_{in}$,
- p_{ref} is such that $^\circ(p_{ref}) = t_{out}$ and $(p_{ref})^\circ = \emptyset$.

Note that with the addition of the place p_{cap}, G is strongly connected with a single cycle.

Example 4: Consider again Example 1 and the structure function detailed in Fig. 1. The TPN model of this system is reported in Fig. 2-left. Observe that the set of operations $\boldsymbol{O} = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$is represented by the set of transitions $\boldsymbol{T} = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ with $t_{out} = t_6$ and $t_{in} = t_7$. The precedence constraints defined with the function F are encoded in the structure of the net. The places p_{exec}, p_{cap} and p_{ref} are connected as specified.

4.2 Resources and supervisor

As previously mentioned, a set of resources R_j is generally required to perform each operation o_j. Each type of resource $r_k \in R_j$ is represented by a specific places p_{rk} with an initial marking that indicates how many resources of type k are available to perform the different operations. More precisely, for each type of resources required to perform the job, a place p_{rk} is added in $\boldsymbol{P}$ with $^\circ(p_{rk}) = (p_{rk})^\circ = \{ t_j$ such that $r_k \in R_j \}$. A row vector of size q is added to the matrices W_{PR} and W_{PO} and $W_{PO}(k,j) = W_{PR}(k,j) = 1$ for all $t_j \in {}^\circ(p_{rk})$ (the others entries of row k being 0).

Supervisory control approach can be adopted to manage resources in order to avoid deadlocks. In particular, the synthesis of a set of monitor places turns out to be very useful for the considered scheduling problem. A monitor place is a place suitable connected with input and output transitions of resource places so that the acquisition and the release of each resource can be

managed to avoid deadlocks. Indeed, each monitor place acts as a supervisor since it represents an addition precondition for the enabling of transitions which is connected to. The firing of a transition yielding a deadlock state results to be disabled by monitor places. As far as ordinary nets are considered, for each monitor place $p_{ck} \in \boldsymbol{CONT}$, a row vector of size q is added to the matrices W_{PR} and W_{PO} with entries equal to 1 or 0. The synthesis of monitor places to avoid deadlock is outside the scope of this paper and the reader can refer to [5], [9], but it is important to notice that "supervised" system is still modeled by a PN.

In order to model a structured FMS with a set of jobs $\boldsymbol{J}$, one design first the model of each structured job of the FMS, then aggregate the different models. Finally, the reference place p_{ref}, the common resource places p_{rk} and the common monitor places p_{ck} are merged (as far as they are used by several jobs). Observe that the resulting global model is no longer a nested structure (due to the merging of the places p_{ref}, p_{rk} and p_{ck}) but this global model is still composed by several subnets each of them being a nested structure. As explained in Section VI, in particular with Proposition 5, the proposed approach is fully applicable for structured FMS.

Example 5: Fig. 7 is the TPN model of a structured FMS with two jobs that have no intermediate buffers. The job J_1 is a structured job with 7 operations o_1 to o_7 and a choice between the sequences of operations $\{o_1,\ldots,o_5\}$ and $\{o_6, o_7\}$. The lot size for J_1 is: $cap_1 = 1$ and $n_1 = 5$ is the number of times this job needs to be performed. The job J_2 is also a structured job with 5 operations o_8 to o_{12} and is structured with a parallelism between the sequences of operations $\{o_8,\ldots,o_{11}\}$ and $\{ o_8, o_{12}, o_{11} \}$. The lot size and number of executions for J_2 are: $cap_2 = 1$ and $n_2 = 3$. The two jobs need 5 resources p_{r1} to p_{r5} (represented in red in Fig. 7). Table 1 details the durations and resources for each operation. This model (without considering the supervisor in blue) has 502 reachable markings and 31 deadlocks when $n_1 = 5$ and $n_2 = 3$ that occur when the resources are reserved for the jobs 1 and 2 in an unappropriated way. A supervisor can be computed for this system [27] in order to avoid all deadlocks, excepted the deadlock that corresponds to the final marking $M(p_{ref}) = n_1 + n_2 = 8$ where both jobs have been performed the expected number of times. This supervisor consists of 2 monitor places p_{c1} and p_{c2} (represented in blue in Fig. 7). The whole model with the supervisor has 352 reachable markings.

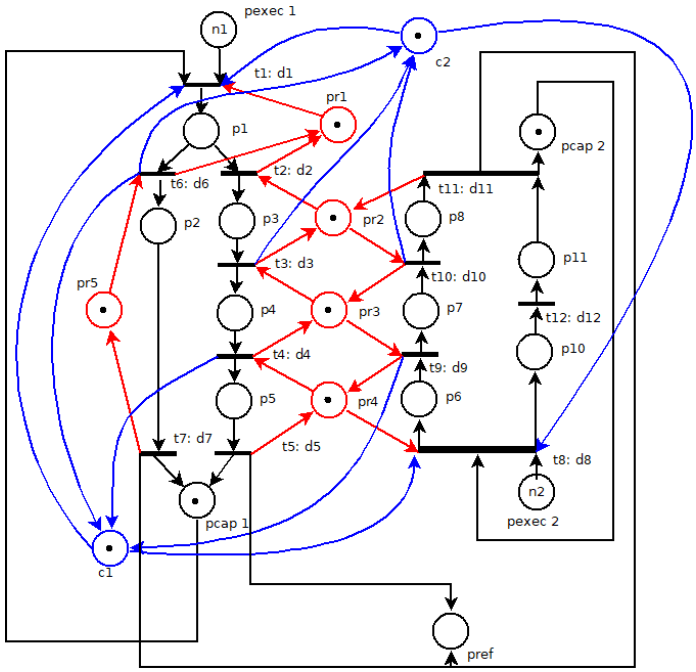

Fig.7: TPN model of the structured FMS on Example 5 with shared resources and monitor places

Table 1: Operations and resources for jobs J_1 and J_2

o_i	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9	o_{10}	o_{11}	o_{12}
t_i	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_5	t_6	t_7	t_8	t_9	t_1
d_i	1TU	1TU	1TU	1TU	1TU	1TU	1TU	1TU	1TU	1TU	1TU	1TU
R_i	r_1	r_1, r_2	r_2, r_3	r_3, r_4	r_4	r_1, r_5	r_5	r_4	r_3, r_4	r_2, r_3	r_2	

Observe that the complexity of a scheduling problems relies on different parameters including:

- the sizing parameters: in particular, the number of jobs and the number of operations in each job,
- the constraints due to the lot size of the jobs: how many products can circulate simultaneously in the same job,
- the constraints due to the resources: how many types of resource are shared, how many resources of each type are available and how many operations need each type of resource,
- the constraints due to the supervisor: how many monitor places are used and how many monitor tokens circulate in the net.
- the objective: how many times each job should be performed.

For simplicity, each job of the considered FMS is assumed to have a lot size of 1 in this paper. Consequently, making abstraction of the places p_{exec}, p_{ref}, p_{rk} and p_{ck} the other places of the model are 1-bounded.

5. A TREE STRUCTURE TO REPRESENT THE TEMPORAL INFORMATION

In this section, the ST encoding is first introduced, then it is used also to encode the temporal information of a structured TPN. In particular, an important property of the structure trees is that their execution provides a precise estimation of a forthcoming sequence of operations based on the list of these operations (making abstraction of the order of these operations). More precisely, let us consider a feasible firing sequence σ, executed in a given structured TPN, under earliest firing policy and assume that only its firing count vector $X(\sigma)$ is known (the sequence σ is unknown by itself). Next, we detail how to compute an interval of the duration $d(\sigma)$ of σ thanks to the ST tree of the net.

5.1 ST tree of structured PNs

A structured PN may be encoded in a systematic way with a tree named *ST* that is designed by successive transformations of the original PN structure [22]. At each iteration, some places are removed and two transitions are summed up in a single one. Each node *j* of *ST* corresponds either to a transition of the original net or to a nested structure previously obtained. A node has 5 attributes:

- *ST*{*j*}.*label*: the index *j* of the node,
- *ST*{*j*}.*type*: S for an elementary sequence, C for an elementary choice, P for a an elementary parallelism, *j* for the terminal node that represents the transition t_j,
- *ST*{*j*}.*first*: the index of the first successor of node *j* in *ST*,
- *ST*{*j*}.*second*: the index of the second successor of node *j* in *ST*,
- *ST*{*j*}.*pred*: the index of the unique predecessor of node *j* in *ST*. *ST*{*j*}.*pred* is empty if *j* has no predecessor

Algorithm 1: **ST tree design**

(Inputs: W_{PR}, W_{PO}, $\boldsymbol{T}$ Output: ST)

1. initialization: $\boldsymbol{E} \leftarrow \boldsymbol{T}$, $k \leftarrow q+1$
2. define the q first nodes of ST according to the transitions in $\boldsymbol{T}$
3. while $|\boldsymbol{E}| > 1$
4. sort $\boldsymbol{E}$ by indexes and call $N(1)$ the node in position 1 in $\boldsymbol{E}$
5. while N' exists within $\boldsymbol{E} - \{ N(1) \}$ such that $\{ N(1), N' \}$ is an elementary sequence
6. create the new node $N(k)$ with type S, label k and successors *first* and *second* are respectively $N(1)$ and N'
7. define the predecessor of $N(1)$ and N' as $N(k)$
8. remove $N(1)$ and N' from $\boldsymbol{E}$ and add $N(k)$ to $\boldsymbol{E}$
9. $k \leftarrow k+1$
10. end while
11. call $N(1)$ the node in position 1 in $\boldsymbol{E}$
12. while N' exists within $\boldsymbol{E} - \{ N(1) \}$ such that $\{ N(1), N' \}$ is an elementary choice or parallelism
13. create the new node $N(k)$ with type C or P, label k, and successors $N(1)$ and N'
14. define the predecessor of $N(1)$ and N' as $N(k)$
15. remove $N(1)$ and N' from $\boldsymbol{E}$ and add $N(k)$ to $\boldsymbol{E}$
16. $k \leftarrow k+1$
17. call $N(1)$ the (new) node in position 1 in $\boldsymbol{E}$
18. end while
19. end while

Proposition 1 states the existence of a tree that encodes any structured PN. The proof is constructive and explains also Algorithm 1. Note that Algorithm 1 also provides a test to check if a given PN is structured.

Proposition 1: Let $G = \langle \boldsymbol{P}, \boldsymbol{T}, W_{PR}, W_{PO} \rangle$ be a PN with q transitions and let ST be the tree obtained with Algorithm 1 for ($\boldsymbol{T}$, W_{PR}, W_{PO}). G is structured if and only if ST has a single node j such that $ST\{j\}.pred = \emptyset$ (i.e, without any predecessor).

Proof: The proof is based on iterative transformations of ($\boldsymbol{T}$, W_{PR}, W_{PO}) in order to design ST. The transformations are organized in a set of iterations and each iteration has two stages. At first iteration, the first stage is to reduce all elementary sequences by removing the intermediate places p_i and summing up the upstream and downstream transitions t_{j1} and t_{j2} in a single transition t_S (Fig. 4a). For each elementary sequence, a node is also added in ST that is tagged with value S (as Sequence) and attributes *first* and *second* are defined with t_{j1} and t_{j2} respectively. When the net contains sequences with $k > 2$ transitions, then k-1 elementary sequence reductions are successively performed. At the end of this process, the resulting net structure has no longer any sequence and is composed only of nested choices and parallelisms.

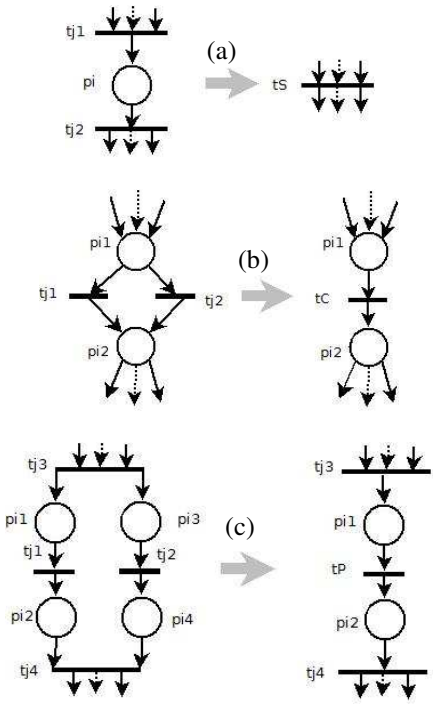

Fig. 4: Elementary structures reduction: sequence (a); choice (b); parallelism (c)

The second stage is to reduce the innermost nested elementary choices and parallelisms. In the case of elementary choices, the transitions t_{j1} and t_{j2} are summed up in a single transition t_C and no place is removed (Fig. 4b). In the case of elementary parallelisms, the places p_{i3} and p_{i4} are removed and the transitions t_{j1} and t_{j2} are summed up in a single transition t_P (Fig. 4c). In both cases, a node is also added in *ST* that is tagged either with value C (as Choice) or P (as Parallelism) and successors t_{j1} and t_{j2} (for choices and parallelism the affectation of t_{j1} and t_{j2} to *first* and *second* does not matter). When the net contains choices (resp. parallelisms) with $k > 2$ branches, then k-1 elementary C (resp. P) reductions are successively performed. Due to the previous transformations, the resulting PN structure has less transitions than the original one and the depth of nested structures is decreased by 1. Thus the same rules can be applied at iteration 2 and the transformations continue in a similar way. The net structure decreases in number of transitions and in the same time *ST* increases in number of nodes. As far as *G* is structured, the transformations can go on to result finally, after q-1 transformations (q is the number of transitions in the net) in a simplified net structure with only one transition. On the contrary, if the net is not structured, then the transformations stop at a given iteration after a number of transformations that is strictly less than q-1 and the final tree has more than one node j that satisfies $ST\{j\}.pred = \varnothing$.

Corollary 2: The complexity of the ST design in number of transformation is at most q-1 where q is the number of transitions in the structured PN model.

Proof: From Fig. 4, one can notice that the application of each reduction rule removes exactly one transition from the original structure and adds one node in the ST tree. If the ST design ends with a single node j that satisfies $ST\{j\}.pred = \varnothing$, q-1 successive reductions are required to build the whole tree. Note that these q-1 successive transformations are organized in a number of iterations in range [1, q-1] depending on the structure of the original net. If the ST design ends with several nodes j that satisfy $ST\{j\}.pred = \varnothing$, then the number of successive transformations is strictly less than q-1.

Note that the ST tree of a given structured net may be enforced to be a unique construction by specifying an order within the nodes of the tree (see Annex 1 for a detailed proof).

Example 3: Consider, the PN in Fig. 2-right. This net is a structured PN with $t_{in} = t_7$ and $t_{out} = t_6$. The application of Algorithm 1 leads to the tree of Fig. 5 (for readability the successor with attribute *first* is always represented as the left branch and the successor with attribute *second* is always represented as the right branch). The successive transformations of the PN structure require 3 iterations. The nodes with indexes 1 to 7 encode the transitions t_1 to t_7 and are considered in $\boldsymbol{E_1}$ to start the design. The elementary sequences {1, 2} and {3, 4} are consecutively encoded with S nodes of indexes 8 and 9. At the end of this process $\boldsymbol{E_1}$ = {5, 6, 7, 8, 9}. Then, node 10 encodes the choice {8, 9}. Finally, at the end of iteration 1, $\boldsymbol{E_2}$ is initialized with {5, 6, 7, 10}. No elementary sequence reduction is detected at iteration 2 but node 11 encodes the parallelism {5, 10}. Consequently, 5 and 10 are removed from $\boldsymbol{E_2}$ and $\boldsymbol{E_3}$ is initialized with {6, 7, 11}. Finally, the S node 12 encodes the sequence {6, 11}, the S node 13 encodes the sequence {12, 7} and the construction ends.

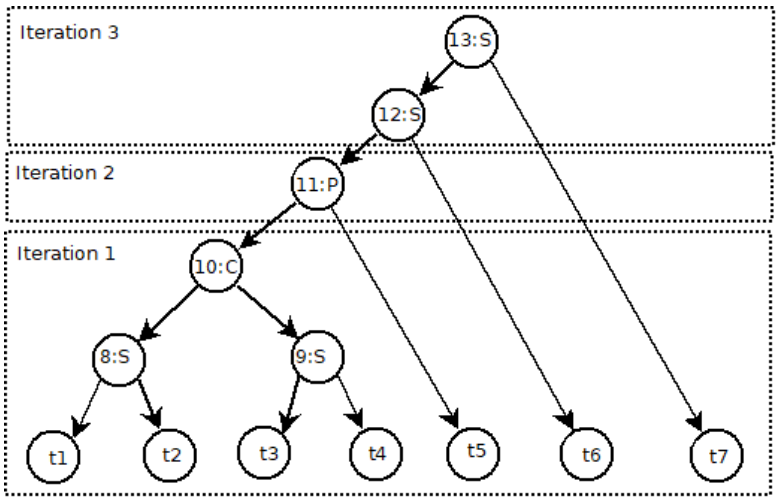

Fig. 5. *ST* tree that encodes the structured PN in Fig. 2-right

5.2 Encoding the temporal information of a structured TPNST

In [22], the authors have started a study to approximate the minimal duration $d(\sigma)$ from the ST tree encoding and the firing count vector $X(\sigma)$. They have also noticed that for some specific structured TPNs it is not possible to compute the exact duration of a sequence from the knowledge of its firing count vector only, (see [22] for more details). Consequently, an estimation of $d(\sigma)$ is defined as an interval to which $d(\sigma)$ necessarily belongs. Starting from the entries of the firing count vector, and using *D* and *ST*, it is possible to evaluate a time interval for one complete execution of any nested structure and to calculate another time interval if the nested structure is partially executed. Five additional attributes are considered for each node of *ST*:

- $ST\{j\}.x$: number of executions of node *j*,
- $ST\{j\}.d_{inf}$ and $ST\{j\}.d_{sup}$: respectively the minimal and maximal durations to execute the node *j,*
- $ST\{j\}.r_{inf}$ and $ST\{j\}.r_{sup}$ respectively the minimal and maximal residual durations when the node *j* is partially executed.

Finally, the duration *d(j)* of the execution of the node *j* belongs to the time interval defined by (2):

$$d(j) \in [ST\{j\}.x \times ST\{j\}.d_{inf} + ST\{j\}.r_{inf} : ST\{j\}.x \times ST\{j\}.d_{sup} + ST\{j\}.r_{sup}\ ] \quad (2)$$

The values of attributes $ST\{j\}.x$, $[ST\{j\}.d_{inf} : ST\{j\}.d_{sup}]$ and $[ST\{j\}.r_{inf} : ST\{j\}.r_{sup}]$ are successively computed from the terminal nodes to the root node of ST. The process propagates $ST\{j\}.x$ and $[ST\{j\}.d_{inf} : ST\{j\}.d_{sup}]$ from the terminal nodes to the root node. The residual durations of the partially executed structures are also propagated with $[ST\{j\}.r_{inf} : ST\{j\}.r_{sup}]$. The minimal duration $d(\sigma)$ belongs to an interval obtained with Proposition 3. The proof is constructive.

Proposition 3: Consider a structured TPN that behaves under earliest firing policy, a feasible sequence σ of duration $d(\sigma)$ and $X(\sigma) = X = (x(t_j))_{j=1,\ldots,q}$. Then $d(\sigma)$ satisfies:

$$d(\sigma) \in [ST\{j_0\}.x \times ST\{j_0\}.d_{inf} + ST\{j_0\}.r_{inf} : ST\{j_0\}.x \times ST\{j_0\}.d_{sup} + ST\{j_0\}.r_{sup}\ ] \quad (3)$$

when $ST\{j\}.x = x(t_j)$ for the terminal nodes j, $j=1\ldots q$ of ST and j_0 is the unique node that satisfies $ST\{\ j_0\ \}.pred = \varnothing$.

Proof: The terminal nodes j, $j=1\ldots q$ of ST are first tagged according to $ST\{j\}.x = x(t_j)$, $ST\{j\}.d_{inf} = ST\{j\}.d_{sup} = d_{min\ j}$ and $ST\{j\}.r_{inf} = ST\{j\}.r_{sup} = 0$. Then, the attributes $ST\{j\}.x$, $ST\{j\}.d_{inf}$, $ST\{j\}.d_{sup}$ and $ST\{j\}.r_{inf}$, $ST\{j\}.r_{sup}$ propagate from the terminal nodes to the root node with a systematic exploration of ST that satisfies all specifications of the TPN time semantics.

- For a S-node j with successors j_1 and j_2:

$ST\{j\}.d_{inf} = ST\{j_1\}.d_{inf} + ST\{j_2\}.d_{inf}$

$ST\{j\}.d_{sup} = ST\{j_1\}.d_{sup} + ST\{j_2\}.d_{sup}$

$ST\{j\}.x = \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\}$

$ST\{j\}.r_{inf} = (ST\{j_1\}.x - \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\}) \times ST\{j_1\}.d_{inf} + (ST\{j_2\}.x - \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\}) \times ST\{j_2\}.d_{inf} + ST\{j_1\}.r_{inf} + ST\{j_2\}.r_{inf}$

$ST\{j\}.r_{sup} = (ST\{j_1\}.x - \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\})\ ST\{j_1\}.d_{sup} + (ST\{j_2\}.x - \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\}) \times ST\{j_2\}.d_{sup} + ST\{j_1\}.r_{sup} + ST\{j_2\}.r_{sup}$

- For a C-node j with successors j_1 and j_2:

$ST\{j\}.d_{inf} = \min\{\ ST\{j_1\}.d_{inf}, ST\{j_2\}.d_{inf}\}$

$ST\{j\}.d_{sup} = \max\{\ ST\{j_1\}.d_{sup}, ST\{j_2\}.d_{sup}\}$

$ST\{j\}.x = ST\{j_1\}.x + ST\{j_2\}.x$

$ST\{j\}.r_{inf} = ST\{j_1\}.r_{inf} + ST\{j_2\}.r_{inf}$

$ST\{j\}.r_{sup} = ST\{j_1\}.r_{sup} + ST\{j_2\}.r_{sup}$

- For a P-node j with successors j_1 and j_2:

$ST\{j\}.d_{inf} = \max\{\ ST\{j_1\}.d_{inf}, ST\{j_2\}.d_{inf}\}$

$ST\{j\}.d_{sup} = \max\{\ ST\{j_1\}.d_{sup}, ST\{j_2\}.d_{sup}\}$

$ST\{j\}.x = \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\}$

$ST\{j\}.r_{inf} = \max((ST\{j_1\}.x - \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\}) \times ST\{j_1\}.d_{inf}, (ST\{j_2\}.x - \min\{\ ST\{j_1\}.x, ST\{j_2\}.x\}) \times ST\{j_2\}.d_{inf}, ST\{j_1\}.r_{inf}, ST\{j_2\}.r_{inf})$

$ST\{j\}.r_{sup} = \max((ST\{j_1\}.x - \min\{ST\{j_1\}.x, ST\{j_2\}.x\}) \times ST\{j_1\}.d_{sup}, (ST\{j_2\}.x - \min\{ST\{j_1\}.x, ST\{j_2\}.x\}) \times ST\{j_2\}.d_{sup}, ST\{j_1\}.r_{sup}, ST\{j_2\}.r_{sup})$

Notice that the duration estimation for a P-node takes into account the effects of interleaving and filters it. Moreover, in a single step it adds to the tree the timed information essential to the estimate of the sequence duration.

The minimal duration $d(\sigma)$ required to execute a feasible sequence σ consistent with X satisfies $d(\sigma) \geq ST\{j_0\}.x \times ST\{j_0\}.d_{inf} + ST\{j_0\}.r_{inf}$ and $d(\sigma) \leq ST\{j_0\}.x \times ST\{j_0\}.d_{sup} + ST\{j_0\}.r_{sup}$.

Example 6: Consider again the TPN of Fig. 2-left with $D = (2\ 4\ 0\ 7\ 9\ 8\ 7)^T$, its *ST* tree reported in Fig. 5, the sequence $\sigma_1 = t_7t_1t_2\,t_5t_6t_7t_3t_4t_5t_6$ and $X_1 = X(\sigma_1) = (1\ 1\ 1\ 1\ 2\ 2\ 2)^T$. Table 2 illustrates the determination of the estimation $d(\sigma_1)$ with Proposition 3. The iterations refer to Fig. 5. The final approximation, obtained at iteration 3, leads to $d(\sigma_1) \in [48 : 48]$. In the present case, one can conclude that $d(\sigma_1)$ =48 TU. But, as mentioned previously, the estimation does not necessarily coincide with the exact value: consider again the same example of TPN with $D' = (0\ 4\ 0\ 3\ 0\ 5\ 0)^T$. For $X = (1\ 1\ 1\ 1\ 2\ 2\ 2)^T$ the true duration of all sequences consistent with X is 17 TU and Proposition 3 leads to the estimation $d(\sigma_1) \in [16 : 18]$.

Table 2: Estimation of $d(\sigma_1)$ with Proposition 3

Iteration	$ST\{j\}.x$	$ST\{j\}.d_{inf}$	$ST\{j\}.d_{sup}$
1	$ST\{j\}.x = 1,\ j = 1,\ldots,4$ $ST\{j\}.x = 2,\ j = 5, 6, 7$	$ST\{j\}.d_{inf} = d_j,\ j = 1,\ldots,7$	$ST\{j\}.d_{sup} = d_j,\ j = 1,\ldots,7$
	$ST\{8\}.x = 1$	$ST\{8\}.d_{inf} = 6$	$ST\{8\}.d_{sup} = 6$
	$ST\{9\}.x = 1$	$ST\{9\}.d_{inf} = 7$	$ST\{9\}.d_{sup} = 7$
	$ST\{10\}.x = 2$	$ST\{10\}.d_{inf} = 6$	$ST\{10\}.d_{sup} = 7$
2	$ST\{11\}.x = 2$	$ST\{11\}.d_{inf} = 9$	$ST\{11\}.d_{sup} = 9$
3	$ST\{12\}.x = 2$	$ST\{12\}.d_{inf} = 17$	$ST\{12\}.d_{sup} = 17$
	$ST\{13\}.x = 2$	$ST\{13\}.d_{inf} = 24$	$ST\{13\}.d_{sup} = 24$

6. SCHEDULING OF STRUCTURED FMSs

In this section we consider the problem to optimize the makespan for structured FMSs. The method is based on a variant of the BS algorithms - namely Hybrid Filtered Beam Search (HFBS) algorithm - that prevents the combinatory explosion. This method is an exploration algorithm that decides the regions to be explored in the reachability graph according to a cost function that is used to sort and select the markings of interest.

6.1 HFBS exploration algorithm

The HFBS algorithm is one improved variant of A^* methods. This algorithm searches for sequences of minimal cost from an initial marking to a reference one. HFBS Algorithm uses a closed list of candidates *OPEN* as the key structure to model the

regions to be explored. The main idea behind this method is, at each iteration, to select (and then remove) the best candidate in *OPEN*, to compute its successors, then add these successors in the list *OPEN*, and to sort the candidates in *OPEN* according to a given cost function (f). What is called here an iteration is the expansion of a given candidate. The list *OPEN* is updated each time a candidate is expanded. Each candidate $S = (M, \sigma, CAL, g, h)$ is a 5-tuple:

- a marking M,
- the sequence σ such that $M_I [\sigma > M$,
- the calendar $CAL(M)$ of the remaining firing times for all transitions enabled at M,
- the duration g for the already done trajectory from M_I to M,
- an estimation h of the residual time from M to the reference marking M_{ref}.

The list *OPEN* saves only the β_g best candidates for the next iteration. β_g is an input parameter to be tuned. Consequently, at each iteration, some candidates are lost and convergence to the optimal solution is, in general, not ensured, but the numerical complexity is improved in space and time compared to the A* Algorithm because the size of the list *OPEN* does never exceed β_g. The HFBS uses not only a global filter with parameter β_g but also a local filter that selects and saves only the β_l best successors, for each expanded candidate. This variant is motivated in order to ensure a better variability in the population of candidates by limiting the number of successors issued from the same parent.

The cost function f is basically composed by the duration of the already performed trajectory (g) plus an estimation of the residual duration (h) to the reference. The functions g and h will be detailed respectively in Sections 6.2 and 6.3. The cost function f is required to sort the candidates in the lists *OPEN* and consequently plays a central role in the efficiency of the method. This cost function is formally defined for each marking M of a given trajectory (σ,M_I) that aims to reach the reference marking M_{ref}. For this purpose (σ,M_I) is divided into two parts: the already computed trajectory (σ_1,M_I) from M_I to M (i.e. $M_I [\sigma_1 > M$) and the unknown residual trajectory (σ_2,M_I) from M to M_{ref} (i.e. $M [\sigma_2 > M_{ref}$). The cost function is written as in (6):

$$f(M_I, M, M_{ref}) = g(\sigma_1,M_I) + h(M,M_{ref}) \quad (6)$$

The function $g(\sigma_1,M_I)$ gives the duration of the already computed trajectory and the heuristic function $h(M,M_{ref})$ estimates the residual duration of (σ_2,M). The candidates in the list *OPEN* are sorted by ascending order of their f values. For a given value of the function f (and in particular for the smallest value of the function f) several candidates may exist in *OPEN*. These candidates could be also sorted in a second round by descending order of their g value (and this will lead to a depth-first-exploration) or alternatively, they could be also sorted by ascending order of their g value (and this will lead to a width-first-exploration). The efficiency of both variants depends on the scheduling problem.

Algorithm 2 that encodes the HFBS method for TPNs, uses the initial marking M_I, the reference marking M_{ref} and the parameters β_g and β_l as inputs and returns the flag *success* = 1 if it finds a solution, otherwise it returns *success* = 0. It also returns the control sequence σ^* of minimal cost and its duration f^* (as found by Algorithm 2)

Algorithm 2: Hybrid Filtered Beam Search

(Inputs: $< G, M_I, D >$, β_g, β_l, M_I, M_{ref}; Outputs: *success*, σ^*, f^*)

1. initialization: $h \leftarrow h(M_I,M_{ref})$, $OPEN \leftarrow \{(M_I, \varepsilon, CAL(\varepsilon,M_I), 0, h)\}$, $\sigma^* \leftarrow \emptyset$, $f^* \leftarrow \infty$, *success* $\leftarrow 0$
2. while $(OPEN \neq \emptyset) \wedge (\ success = 0)$

3. remove the first candidate $S = (M, \sigma, CAL, g, h)$ in *OPEN*
4. if $M = M_{ref}$
5. $success \leftarrow 1$, $\sigma^* \leftarrow \sigma, f^* \leftarrow g$
6. else
7. if $CAL \neq \emptyset$
8. $TEMP \leftarrow \emptyset$
9. for each t enabled at M
10. compute M' such that $M [t > M'$
11. $\sigma' \leftarrow \sigma t$, $CAL' \leftarrow CAL(\sigma', M')$
12. $g' \leftarrow g(\sigma', M_I)$, $h' \leftarrow h(M', M_{ref})$
13. $TEMP \leftarrow TEMP \cup \{(M', \sigma', CAL', g', h')\}$
14. end for
15. sort *TEMP* by ascending value of $g+h$
16. select the β_l first candidates of *TEMP* and add them in *OPEN*
17. end if
18. sort *OPEN* by ascending value of $g+h$
19. save the β_g first candidates of *OPEN* and remove the other candidates
20. end if
21. end while

The HFBS algorithm may be viewed as a fast variant of A* algorithm and is motivated because A* is unfortunately not tractable for large systems due to the unlimited increase of the list *OPEN*. But the HFBS algorithm may present degraded performance compared to A*. Observe that A* Algorithm will always find a solution (as far as one solution exists) provided the distance between two different nodes is strictly positive and this is no longer true with the HFBS that may fill the list *OPEN* with forbidden markings (i.e., markings that will necessary lead to deadlock situations in the future). Another interesting property of A* algorithm is that when the heuristic function h is admissible, meaning that it never overestimates the actual minimal cost to reach the goal, then A* Algorithm will return the solutions that optimize the cost function. This is also no longer true with the HFBS that may fill the list *OPEN* with candidates that will not necessarily lead to an optimal solution.

6.2 Computation of *g* cost function

As long as an earliest firing policy is applied, Algorithm 3 [24] computes the duration $g(\sigma_1, M_I)$ of (σ_1, M_I) and generates the calendar *CAL* with the remaining durations of the next possible firings. This algorithm uses the chronological firing order of the transitions in σ_1 and the earliest firing policy to compute the firing times of the transitions in σ_I and update the remaining durations of the transitions already enabled at M. These transitions and their remaining firing durations are saved in *CAL*. More formally, for each marking M, obtained by firing a sequence σ at M_I let us define $CAL(\sigma, M) = \{(t, \delta), t \in (M)^\circ\}$ where $(M)^\circ$ is the set of transitions enabled at M, t is a given transition in set $(M)^\circ$ and δ is the remaining duration to fire t. The calendar is initialized assuming that the trajectory starts at date 0 and that no transition is enabled before 0.

Algorithm 3: Timing a sequence σ and generating the calendar *CAL*

(Inputs: $< G, M_I, D >$, σ, M ; Outputs: CAL, σ')

1. initialization: $\tau \leftarrow 0$; $CAL \leftarrow \{(t_j, d_j)$ such that $M [t_j >\}$, $\sigma' \leftarrow (\varepsilon,0)$, $h \leftarrow |\sigma|$, $\tau \leftarrow 0$
2. for k from 1 to h
3. find in CAL the smallest remaining time δ_k of the k^{th} transition $t(j_k)$ in σ
4. $\tau \leftarrow \tau + \delta_k$, remove entry $(t(j_k), \delta_k)$ in CAL
5. $CAL_{new} \leftarrow \emptyset$, $M' \leftarrow M - W_{PR}.X(t(j_k))$
6. for all t' such that $M' [t' >$
7. compute the enabling degree $n'(t',M')$ of t' at M'
8. for j from 1 to $n'(t',M')$
9 find the j^{th} occurrence (t', δ'_j) of t' in CAL
10. $CAL_{new} \leftarrow CAL_{new} \cup (t,' \max(0, \delta'_j - \delta_k))$
11. end for
12. end for
13. $M'' \leftarrow M' + W_{PO}.X(t(j_k))$
14. for all t'' such that $M'' [t'' >$
15. compute the enabling degree $n''(t'',M'')$ of t'' at M''
16. for j from 1 to $n''(t'',M'') - n'(t',M')$
17. $CAL_{new} \leftarrow CAL_{new} \cup (t'', d(t''))$
18. end for
19. end for
20. $CAL \leftarrow CAL_{new}$, $\sigma' \leftarrow \sigma' (t(j_k), \tau)$, $M \leftarrow M''$

21.end for

6.3 Computation of *h* cost function

The estimation $h(M,M_{ref})$ of the residual duration is the core of the HFBS algorithm. As far as the list of candidates is of limited size, the use of an accurate approximation of the residual duration is of particular importance. This choice will not only influence the time the algorithm needs to converge to a solution but it will also influence the cost of the solution and even the success of the search. To be efficient, the estimation $h(M,M_{ref})$ should have two characteristics: (1) it should never overestimate the true residual duration to reach the reference marking, otherwise it may eliminate good candidates; (2) the difference between the estimation and the true duration should be as small as possible in order to sort the candidates with enough accuracy. Several estimations of the residual duration from the current marking M to the reference M_{ref} have been studied, and numerous are based on study of paths in the PN structure. In particular, Luo in [31] proposed an heuristic function based on the search of resource and operational places for P-timed Petri nets (i.e. timed Petri nets where time is associated to the places). The estimation is then based on the search of the shortest paths from the marked places to a given reference place. In [33] this heuristic was discussed and in [23], the authors adapt the heuristic function to the T-timed Petri nets. Basically, the residual time was approximated with $h_1(M,M_{ref})$ as defined by (7):

$$h_1(M,M_{ref}) = \max\{ \chi^*(p_i, p_{ref}) \text{ such that } p_i \in P(M)\} \quad (7)$$

For TPNs that reserve the tokens in order to fire the enabled transitions, the estimation can be refined with a term that corrects the remaining firing duration of the first transition in each path $pth^*(p_i, p_{ref})$ according to the remaining duration in $CAL(\sigma_1,M)$. This correction prevents to overestimate the duration of a firing that has already started at M but is not completed. One can notice that the correction term is negative because $\delta_j \leq d_j$ and bounded because $\delta_j \geq 0$.

$$h_1(M, M_{ref}) = \max\left\{\left(\chi^*(p_i, p_{ref}) + \sum_{\substack{t_j \in CAL(\sigma_1,M) \\ \cap\, \chi^*(p_i,p_{ref})}} (\delta_j - d_j)\right) \text{ such that } p_i \in P(M)\right\} \quad (8)$$

However, the heuristic function $h_1(M,M_{ref})$ based on paths provides a poor lower bound of the residual time to M_{ref} in numerous structured jobs. The problem occurs when parallelisms exist in some jobs. This is due to the fact that $\chi^*(p_i, p_{ref})$ is defined as the minimal duration over all paths from p_i to p_{ref}. In case of a parallelism, the path of minimal duration may be far from the true duration and for this particular structure one should consider instead the path of maximal duration. In a structured job that may combine several choices and parallelisms, an improved estimation function should be considered. In Section 5, we have shown that a time interval can be computed to estimate the duration of a given firing count vector for structured TPNs that behave under earliest firing policy. In the next, we use this result to propose an improved estimation of the residual time to the reference for structured jobs.

First, the residual firing count vector $X(M, M_{ref})$ from M to M_{ref} can be obtained by solving the following integer linear programming problem:

$$X(M, M_{ref}) = \text{argmin } \{(D)^T \times X: X \in (\mathbf{N})^q \text{ such that } W \times X = (M_{ref} - M)\} \quad (9)$$

Using equation (9) to compute the firing count vector of a possible firing sequence can lead, in some particular cases, to unfeasible solutions (i.e., sequences that are not fireable in the considered PN model). In such cases, the resulting residual time obtained from this vector will be strictly less than the true duration. This estimation is however acceptable because it will not overestimate the true duration.

Second, an heuristic function $h_2(M,M_{ref})$ that estimates the residual duration to the reference is obtained with the ST tree of the structured TPN. Observe that Proposition 3 provides an interval $d(\sigma)$ that includes the true duration of the residual sequence. Consequently, $h_2(M,M_{ref})$ can be defined by (10).

$$h_2(M,M_{ref}) = ST\{j_0\}.x \times ST\{j_0\}.d_{inf} + ST\{j_0\}.r_{inf} + \sum_{\substack{t_j \in CAL(\sigma_1,M) \\ \cap \boldsymbol{T}(X(M,Mref))}} (\delta_j - d_j) \quad (10)$$

where $ST\{j\}.x$, $j=1\ldots q$ satisfies $ST\{j\}.x = x(M, M_{ref}, t_j)$ (i.e. the entry j of vector $X(M, M_{ref})$ given by (9)) and $\boldsymbol{T}(X(M, M_{ref}))$ is the support of $X(M, M_{ref})$. Proposition 4 proves that $h_2(M,M_{ref})$ defined by (10) is a lower bound of the residual duration to the reference marking.

Proposition 4: Consider a structured job with a lot size of 1 modeled with an extended structured TPN. The heuristic function $h_2(M,M_{ref})$ obtained with (10) is a lower bound of the residual duration to the reference marking.

Proof: At a given marking M, the residual firing count vector $X(M, M_{ref})$ to the reference is computed with equation (9). Observe that the remaining time of the transitions t_j that belong to $\boldsymbol{T}(X(M, M_{ref}))$ and that also appear in $CAL(\sigma_1,M)$ may be smaller that d_j. For such transitions, a correction similar to the correction already used in (8) is applied to prevent to overestimate the duration of their firing. As a consequence, $h_2(M,M_{ref})$ is a lower bound of the residual duration.

From the previous, result it becomes possible to compute a lower bound of the residual duration to the reference marking for structured FMSs.

Proposition 5: Consider a structured FMS with K jobs with lot size equal to 1 and the K structured TPNs with their ST trees that model the K jobs. The heuristic function $h_2(M,M_{ref})$ computed with (11) is a lower bound of the residual duration to the reference marking.

$$h_2(M,M_{ref}) = \max\{h^k{}_2(M,M_{ref}) \text{ such that } k = 1,\ldots,K\} \quad (11)$$

with $h^k{}_2(M,M_{ref})$ is computed with (10) for the kth job.

Proof: First, observe that Proposition 4 is applicable to each job J_k and $h^k{}_2(M,M_{ref})$ is a lower bound of the residual duration required to perform job J_k. Second, the estimation $h^k{}_2(M,M_{ref})$ is also a lower bound of the residual duration required to perform all jobs of the FMS, because the resource constraints increase the makespan by slowing down the execution of the jobs. Consequently, $h_2(M,M_{ref})$ computed with (11) is a lower bound of the residual duration to the reference.

In simple words, Proposition 5 states that using the maximal value of the residual durations, computed with (10) for each job, remains a lower bound of the true makespan and, consequently, good candidates are not eliminated. This explains why the method is fully applicable for structured FMS (even if the global TPN model of the FMS is not a structured net).

Example 7: In order to illustrate the performance of the proposed estimation function, compared to the usual estimation function that is based on paths, let us consider again the TPN of Example 2, detailed in Fig. 2-left. Note that the makespan to execute once this job is obtained as $C_{max} = d_6 + d_7 + \max(d, d')$ where d and d' are the times required to execute each branch of the parallelism. Consequently, $d = d_5$ and d' is the minimal time required to execute o_1 then o_2 or o_3 then o_4. Finally, $C_{max} = d_6 + d_7 + \max(d_5, \min(d_1 + d_2, d_3 + d_4)) = 24$ TU (for the operation durations detailed in Example 2). The use of the LMI (9) and the function h_2 provides an exact estimation $h_2(M_I,M_{ref}) = 24$ TU of C_{max} whereas the use of function h_1 leads to an estimation $h_1(M_I,M_{ref}) = d_6 + d_7 + \min(d_5, d_1 + d_2, d_3 + d_4) = 21$ TU. Additional comparisons of the performance obtained with the cost functions $h_1(M,M_{ref})$ and $h_2(M,M_{ref})$ are proposed in Section 7.

7 EXAMPLES

In this section, the proposed approach is applied on two examples. The first example is an FMS that is studied with three different resource configurations without deadlocks. It illustrates the advantage of the proposed cost function $h_2(M,M_{ref})$ compared to the usual cost function $h_1(M,M_{ref})$ based on paths. The second example is an FMS with resources and deadlocks that is studied first without monitor places and then with monitor places that eliminate the deadlocks and some forbidden markings. This example illustrates the combined use of $h_2(M,M_{ref})$ and supervisory control.

Example 8: In order to validate the proposed approach, a set of 100 different structured FMSs with a common structure is designed (Fig. 8). Each FMS is composed of 2 jobs. The first job J_1 is composed of 11 operations $\{o_1,\ldots,o_{11}\}$ that are structured with 3 sequences $S_1 = \{o_1,\ldots,o_5\}$, $S_2 = \{o_6, o_7\}$, and $S_3 = \{o_8,\ldots,o_{11}\}$, one parallelism P between S_2 and S_3 and finally one sequence between S_1 and P whereas the second job J_2 is composed of a simple sequence $S_4 = \{o_{12},\ldots,o_{16}\}$ with 5 operations. The durations of the operations are randomly chosen in the range [0 : 10] TU and are different for each FMS. The lot size of each job is assumed to be equal to 1 and the objective is to execute each job once. Three different configurations that correspond to 3 levels of resource constraints are tested:

- Configuration A: FMSs without resource,
- Configuration B: FMSs with 2 resources r_1 and r_2 that are shared by 8 operations: r_1 is required to perform operations o_9 and o_{10} in J_1 and also o_{12} and o_{13} in J_2; r_2 is required to perform operations o_4 and o_5 in J_1 and also o_{15} and o_{16} in J_2. This configuration illustrates the cases when only a few resources are shared and when the resources are shared by a few operations.
- Configuration C: FMSs with 4 resources that are shared by 16 operations: in addition to r_1 and r_2 already defined in configuration B, r_3 is required to perform operations o_{11} in J_1 and also o_{12} and o_{13} in J_2; r_4 is required to perform operations o_2, o_3, o_6 and o_8, in J_1 and also o_{15} and o_{16} in J_2. This configuration illustrates the cases when more resources are shared and when the resources are shared by a lot of operations (Fig. 8, red lines).

The parameters for HFBS are $\beta_g = \beta_l = 10$ and a maximal number of 1000 candidates is expanded before the algorithm stops. Table 3 sums up the results for configurations A, B and C. Four indicators are evaluated for validation and comparison issues:

- the success rate is the percentage of the FMSs for which a control sequence was found before reaching the maximal number of 1000 candidates,
- the optimality rate is the percentage of the FMSs for which an optimal control sequence was found (i.e. a sequence that leads to the minimal makespan),
- the complexity in space is evaluated as the mean number of expanded candidates for a given FMS,
- the complexity in time is the mean computation time required to compute a schedule for a given FMS.

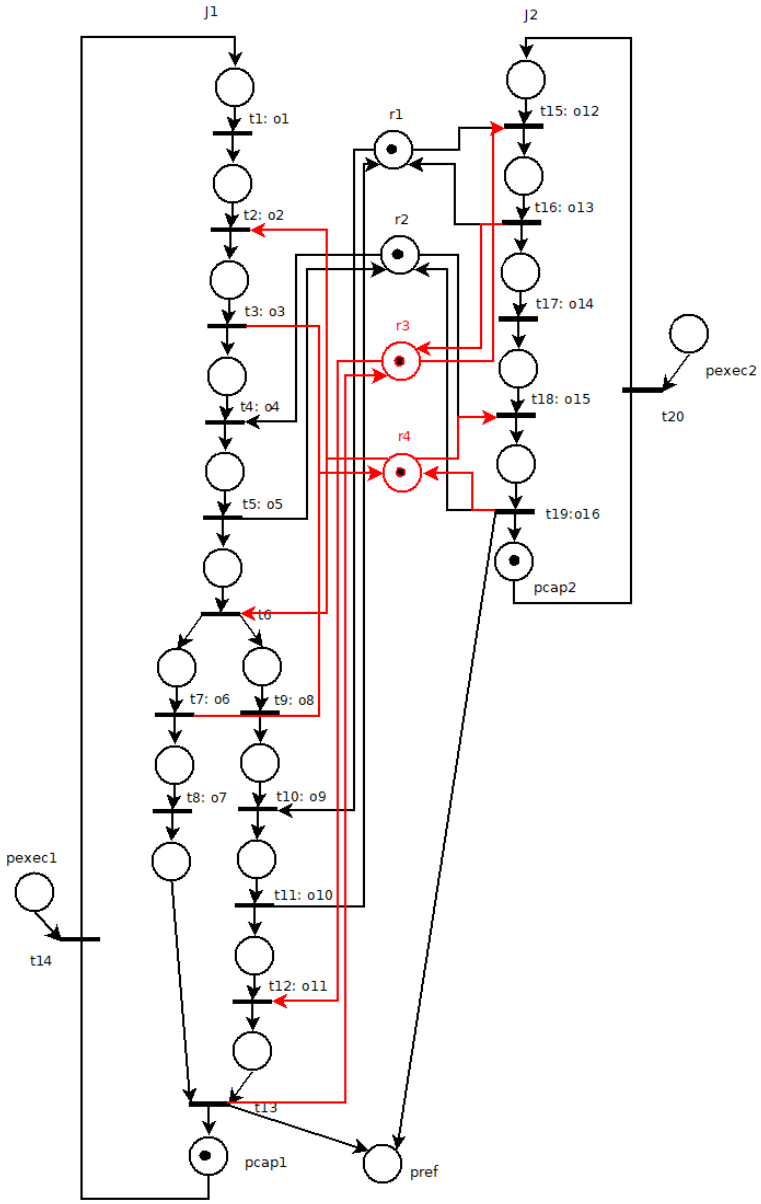

Fig. 8: FMSs with different resource configurations

Considering first the FMS without resource (configuration A). The success and optimality rates are degraded with the use of the heuristic function h_1 compared to h_2 (Table 3). Indeed, with h_1, the performance will depend on the time parameters (that are randomly chosen for the set of considered FMSs). In particular, the durations required to execute the two branches of the parallelism will affect the selection of the best candidates and may prevent the search algorithm to converge onto the optimal solution or even to find any solution. The complexity is space is also from 5 to 20 times larger with h_1 compared to h_2. This excessive space expansion with h_1 is due to the exploration of non-promising candidates. Fig. 9 reports the histograms of the number of candidates with HFBS when the heuristic functions h_1 (in red) and h_2 (in blue) are respectively used. One can notice the strong dispersion of the number of candidates with h_1. Finally, note that the complexity in time is more or less similar with h_1 and h_2 despite the few number of expanded candidates with h_2. Indeed, each candidate expanded with h_2 leads to the resolution of an ILP of the form (9) before evaluating the residual duration with (10) and (11). The resolution of ILP is the stage of the exploration that requires the main part of the computational resource.

Table 3: Results for a set of 100 FMSs for configurations A, B and C

	Configuration	A	B	C
h_1	Success rate	92%	99%	100%
	Optimality	83%	94%	99%
	Complexity in space	479 cand.	276 cand.	322 cand.
	Complexity in time	1.4s	0.6s	0.7s
h_2	Success rate	100%	100%	100%
	Optimality	100%	100%	100%
	Complexity in space	87 cand.	73 cand.	72 cand.
	Complexity in time	3.5s	1.8s	1.8s

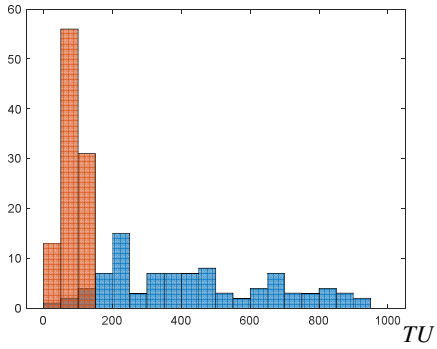

Fig 9: Histogram of the number of candidates for 100 FMSs with HFBS and $\beta_g = \beta_l = 10$

When resources are added to the FMSs, one can notice an improvement of the performances obtained with h_1 (configuration B). In the same time, the performances obtained with h_2 remains optimal. In fact, adding resources in the FMSs restricts the variability of the possible schedules to be explored in order to reach the reference marking. Thus, the sorting and selection of the best candidates are less affected by the poor estimation of the residual time with h_1. The conclusion is similar if the number of resources continues to increase (configuration C).

Example 9: Consider again Example 5 detailed in Fig. 7 and Table 1. A set of instances is studied for this workshop by varying n_1 and n_2 (i.e., the numbers of times that each job should be performed). Table 4 reports the size of the reachability graph without (first line) and with supervisor (second line) with respect to n_1 and n_2. Without supervisor the number of deadlock markings increases to 73 as n_1 and n_2 increase. Observe that the two monitor places (see Fig. 7) eliminate all deadlocks (excepted the deadlock that corresponds to the final marking $M(p_{ref}) = n_1 + n_2$ where both jobs have been performed the expected number of times). Observe also that the number of legal markings accepted by this supervisor is much less than the number of markings reachable in original system (without the addition of the supervisor). Consequently, the minimal processing time C_{max} may increase when the supervisor is used. Tables 5 and 6 report the C_{max} obtained respectively without and with supervisor with respect to n_1 and n_2. Tables 5 and 6 also report the duration of the schedule computed with the HFBS algorithm in both cases. For this purpose, three sets of beam parameters have been tested $(\beta_g, \beta_l) \in \{(5, 2), (10, 4), (20, 4)\}$ and the best results are reported (note that the HFBS method does not necessarily improve the performance by increasing β_g and β_l). The instances for which the computed schedule exceeds C_{max} but with a difference less than 10% are highlighted in grey and the ones for which the difference exceeds 10% are also highlighted in dark grey. The use of a supervisor generally

decreases the difference (even if it increases C_{max} by itself); this is mainly due to the reduction of the number of legal states accepted by the supervisor. Another conclusion is that the complexity of the exploration (that is measured as the number of explored candidates in list *OPEN*) significantly decreases thanks to the use of the supervisor. Consequently, exploration is speeded up thanks to the supervisor; this is not only due to the reduction of the number of legal markings accepted by the supervisor but also because problematic markings are eliminated at first. Without supervisor, such markings may be selected as promising candidates (i.e., candidates with a low value of h_2 function) by the HFBS algorithm at a given iteration and removed latter because they lead to deadlocks. This may slow down the whole exploration.

To conclude, the combined used of HFBS with supervisory control is an interesting strategy to accelerate the convergence of the exploration. The main limitation is due to the specification of the supervisor that generally degrades the makespan. Consequently, application to scheduling issues should be carefully addressed.

Table 4: Number of states and of deadlocks (within parenthesis) without (first line) and with supervisor (second line)

$n_1 \setminus n_2$	1	2	3	4	5	6
1	44 (3) 34 (1)	81 (5) 61 (1)	118 (7) 88 (1)	155 (9) 115 (1)	192 (11) 142 (1)	229 (13) 169 (1)
2	80 (5) 60 (1)	147 (9) 107 (1)	214 (13) 154 (1)	281 (17) 201 (1)	348 (21) 248 (1)	415 (25) 295 (1)
3	116 (7) 86 (1)	213 (13) 153 (1)	310 (19) 220 (1)	407 (25) 287 51°	504 (31) 354 (1)	601 (37) 421 51°
4	152 (9) 112 (1)	279 (17) 199 (1)	406 (25) 286 (1)	533 (33) 373 (1)	660 (41) 460 (1)	787 (49) 547 (1)
5	188 (11) 138 (1)	345 (21) 245 (1)	502 (31) 352 (1)	659 (41) 459 (1)	816 (51) 566 (1)	973 (61) 673 (1)
6	224 (13) 164 (1)	411 (25) 291 (1)	598 (37) 418 (1)	785 (49) 545 (1)	972 (61) 672 (1)	1159 (73) 799 (1)

Table 5: Duration of the schedule computed with of HFBS (first line), C_{max} (within parenthesis) and number of explored candidates in list *OPEN* (second line) without supervisor

$n_1 \setminus n_2$	1	2	3	4	5	6
1	4 (4) 29	8 (8) 58	12 (12) 68	16 (16) 78	20 (20) 88	24 (24) 98
2	6 (6) 2230	8 (8) 1043	12 (12) 135	16 (16) 164	20 (20) 174	24 (24) 184
3	10 (9) 85	9 (9) 12981	12 (12) 2940	16 (16) 3196	20 (20) 3502	24 (24) 3762
4	13 (12) 88	12 (12) 12984	14 (12) 15073	18 (16) 4596	22 (20) 5064	26 (24) 5500
5	16 (15) 91	15 (15) 12987	17 (15) 15075	21 (16) 4599	25 (20) 5067	29 (24) 5503
6	19 (18) 94	18 (18) 12990	20 (18) 15078	24 (18) 4598	28 (20) 5070	32 (24) 5506

Table 6: Duration of the schedule computed with the HFBS (first line), C_{max} (within parenthesis) and number of explored candidates in list *OPEN* (second line) with supervisor

$n_1 \setminus n_2$	1	2	3	4	5	6
1	6 (6) 51	9 (9) 312	14 (13) 93	18 (17) 117	22 (21) 141	26 (25) 165
2	8 (8) 48	11 (11) 238	15 (14) 335	19 (18) 391	23 (22) 446	27 (26) 501
3	11 (11) 54	13 (13) 110	16 (16) 457	20 (19) 528	24 (23) 599	28 (27) 670
4	14 (14) 58	16 (16) 116	18 (18) 528	21 (21) 1489	25 (24) 1922	29 (28) 2058
5	17 (17) 61	19 (19) 122	21 (21) 157	23 (23) 1961	26 (26) 1912	30 (29) 2308
6	20 (20) 64	22 (22) 125	24 (24) 161	26 (26) 201	29 (28) 2087	33 (31) 2331

8 CONCLUSION

Scheduling problems for a class of flexible manufacturing systems including choices and parallelisms between operations, have been considered in this paper. The inclusion of such structures in the different jobs of the FMS deviates the usual heuristic function from an acceptable estimation of the residual time required to reach the reference marking. In order to alleviate the exploration method from this poor estimation, we have proposed a new heuristic function that is based on the ST tree encoding of the TPN models of the jobs. Combined with an ILP approach, the tree encoding is useful to provide a more efficient estimation of the residual duration. As far as deadlock situations exist, the proposed approach can be used directly on the original system without taking care of the deadlocks or in combination with a supervisor that will first reduce the set of legal markings and eliminates the deadlocks. The first option needs generally to enlarge the list of candidates when numerous problematic markings exist. The second option may be considered with attention for scheduling issues as far as the addition of a supervisor generally degrades the makespan.

The numerical effort to build the ST tree has been proved to be linear with respect to the number of transitions and nested structures in the net and also to the number of jobs in the FMS. Consequently, the numerical effort is mainly allocated to the resolution of the ILP and the size of the list of candidates that is handled by the exploration algorithm. In particular, the necessity to solve an ILP for each expanded candidate remains time consuming and may prevent to use the approach for large systems where numerous candidates should be expanded. Consequently, future research will focus on how to combine the path-based heuristic with the ST tree encoding. Instead of computing the residual firing count vector for each expanded candidate, we will compute the shortest paths to the reference by using the tree ST.

REFERENCES

[1] B. Abdallah, H. A. Elmaraghy, and T. Elmekkawy, "Deadlock-free scheduling in flexible manufacturing systems using Petri nets," *Int. J. Prod. Res.*, vol. 40, pp. 2733–2756, 2002.

[2] K. Barkaoui, I. Ben Abdallah, Analysis of a resource allocation problem in FMS using structure theory of Petri nets, *Proceedings, First International Workshop on Manufacturing and Petri Nets*, pp. 1-15, Japan, 1996.

[3] F. Basile, P. Chiacchio, C. Carbone, Feedback control logic for backward conflict free choice nets. IEEE Transactions on Automatic Control, 52(3), 387–400, 2007.

[4] F. Basile, P. Chiacchio, and A. Giua, Suboptimal supervisory control of Petri nets in presence of uncontrollable transitions via monitor places, Automatica, vol. 42, no. 6, pp. 995–1004, 2006.

[5] F. Basile, R. Cordone, and L. Piroddi, "A branch and bound approach for the design of decentralized supervisors in Petri net models," Automatica, vol. 52, pp. 322–333, Feb. 2015.

[6] E. Benveniste, S. Fabre, S. Haar, and C. Jard, Diagnosis of asynchronous discrete-event systems: A net unfolding approach, *IEEE Trans. on Automatic Control*, vol. 48, no. 5, pp. 714–727, 2003.

[7] G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Springer US, 2008.

[8] Y. Chen, Z. Li, K. Barkaoui and M. Uzam, "New Petri Net Structure and Its Application to Optimal Supervisory Control: Interval Inhibitor Arcs," in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 10, pp. 1384-1400, 2014.

[9] R. Cordone, A. Nazeem, L. Piroddi, S. Reveliotis, Maximally permissive deadlock avoidance for sequential resource allocation systems using disjunctions of linear classifiers. In Proc. 51st IEEE conf. on decision and control, pp. 7244–7251, Maui, HI, USA, 2012.

[10] R. David and H. Alla, *Petri nets and grafcet – tools for modelling discrete events systems*, London: Prentice Hall, 1992.

[11] J. Ezpeleta, J.M. Colom, J. Martinez, A Petri Net based deadlock prevention policy for Flexible Manufacturing Systems, *IEEE Trans-actions on Robotics and Automation*, vol. 11, pp. 173-184. 1995.

[12] A. Giua, F. Di Cesare, M. Silva, Generalized mutual exclusion constraints on nets with uncontrollable transitions. In Proc. IEEE int. conf. on systems, man and cybernetics, pp. 974–979, Chicago, IL, USA, 1992.

[13] A. Giua, Supervisory control of Petri nets with language specifications. Lecture Notes in Control and Information Sciences, 433, 235–255, 2013

[14] S. Haddad, P. Moreaux, Stochastic Petri Nets (Chapter 7), in *Petri Nets: Fundamental Models and Applications*, Wiley, 2009.

[15] L. B. Han, K. Y. Xing, X. Chen, H. Lei, and F. Wang, Deadlock-free genetic scheduling for flexible manufacturing systems using Petri nets and deadlock controllers, *Int. J. Prod. Res.*, vol. 52, pp. 1557–1572, 2014.

[16] R. Huang, R. Jiang, and G. Zhang, Search strategy for scheduling flexible manufacturing systems simultaneously using admissible heuristic functions and non-admissible heuristic functions, *Comput. Ind. Eng*., vol. 71, pp. 21–26, 2014.

[17] M. D. Jeng and S. C. Chen, A heuristic search approach using approximate solutions to Petri net state equations for scheduling flexible manufacturing systems, *Int. J. Flexible Manuf. Syst*., vol. 10, no. 2, pp. 139–162, 1998.

[18] B.H. Krogh, L.E. Holloway, Synthesis of feedback control logic for discrete manufacturing systems. Automatica, 27(4), 641–651, 1991.

[19] J. Lee, O. Korbaa O, Modeling and scheduling of ratio-driven FMS using unfolding time Petri nets. *Comput Ind Eng* 46(4):639–653, 2004.

[20] Y. Lee and F. DiCesare, Scheduling flexible manufacturing systems using Petri nets and heuristic search, *IEEE Trans. Robot. Autom.*, vol. 10, no. 2, pp. 123–132, 1994.

[21] Lefebvre, Approaching minimal time control sequences for timed Petri nets, *IEEE Trans. Aut. Science Eng.*, vol. 13, no. 2, pp. 1215-1221, 2016.

[22] Lefebvre D., F. Basile, Design of control sequences for timed Petri nets based on tree encoding, Proc. IFAC WODES 18, Sorrento Coast, Italy, 2018.

[23] D. Lefebvre, G. Mejia, Robust scheduling in uncertain environment with Petri nets and beam search, Proc. IEEE INCOM 2018, Bergamo, Italy, 2018.

[24] D. Lefebvre, C. Daoui, Control design for bounded Partially Controlled TPNs using Timed Extended Reachability Graphs and MDP, *IEEE Trans. Man, Cybernetics and Systems*, early access, DOI: 10.1109/TSMC.2018.2817492, 2018.

[25] H. Lei, K. Xing, L. Han, F. Xiong, Z. Ge, Deadlock-free scheduling for flexible manufacturing systems using Petri nets and heuristic search, *Computers & Industrial Engineering*, vol. 72, pp. 297–305, 2014.

[26] Y. Li, W.M. Wonham, Control of vector discrete-event systems II— controller synthesis. IEEE Transactions on Automatic Control, 39(3), 512–531, 1994.

[27] Z. Li and M. Zhou, Control of Elementary and Dependent Siphons in Petri Nets and Their Application, in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 38, no. 1, pp. 133-148, 2008.

[28] J. Li, M. Zhou and X. Dai, Reduction and Refinement by Algebraic Operations for Petri Net Transformation, in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 42, no. 5, pp. 1244-1255, 2012.

[29] G.Y. Liu, K. Barkaoui, A survey of siphons in Petri nets, *Information Sciences*, vol. 363 pp. 198-220, 2016

[30] G.Y. Liu, P. Li, Z. Li and N. Wu, Robust Deadlock Control for Automated Manufacturing Systems With Unreliable Resources Based on Petri Net Reachability Graphs, in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 7, pp. 1371-1385, 2019.

[31] J.C. Luo, K.Y Xing, M.C. Zhou, X.L. Li, X.N. Wang, Deadlock-Free Scheduling of Automated Manufacturing Systems Using Petri Nets and Hybrid Heuristic Search, *IEEE Trans. Syst. Man and Cyb*., 45(3): 530-541, 2015.

[32] G, Mejía, N. G. Odrey, An approach using Petri nets and improved heuristic search for manufacturing system scheduling, *Journal of Manufacturing Systems*, 24(2), 462-476, 2005.

[33] G. Mejía, K. Nino, A new Hybrid Filtered Beam Search algorithm for deadlock-free scheduling of flexible manufacturing systems using Petri Nets, *Computers & Industrial Engineering*, 108, pp. 165–176, 2017.

[34] G. Mejía, J. P. Caballero-Villalobos and C. Montoya, Petri Nets and Deadlock-Free Scheduling of Open Shop Manufacturing Systems, in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 6, pp. 1017-1028, 2018.

[35] T. Murata, Petri nets: Properties, analysis and applications, in *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, April 1989.

[36] P. S. Ow, T. E. Morton, Filtered beam search in scheduling, *International Journal of Production Research*, 26(1), 35–62, 1988.

[37] C. Pan, M. Zhou, Y. Qiao and N. Wu, Scheduling Cluster Tools in Semiconductor Manufacturing: Recent Advances and Challenges, in *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 586-601, 2018.

[38] Y. Qiao, N. Q. Wu, F. J. Yang, M. C. Zhou, and Q. H. Zhu, Wafer sojourn time fluctuation analysis of time-constrained dual-arm cluster tools with wafer revisiting and activity time variation, IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 48, no. 4, 622-636, 2018.

[39] Y. Qiao, N. Q. Wu, F. J. Yang, M. C. Zhou, Q. H. Zhu, and T. Qu, Robust scheduling of time-constrained dual-arm cluster tools with wafer revisiting and activity time disturbance, IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 49, no. 6, 1228-1240, 2019.
[40] C. Ramchandani, *Analysis of asynchronous concurrent systems by timed Petri nets*, Ph. D, MIT, USA, 1973.
[41] A. Reyes, H. Yu, G. Kelleher, and S. Lloyd, Integrating Petri nets and hybrid heuristic search for the scheduling of FMS, Computers in Industry, vol. 47, no. 1, pp. 123–138, 2002.
[42] T. Sun, C. Cheng, and L. Fu, A Petri net based approach to modeling and scheduling for an FMS and a case study, *IEEE Trans. Robot. Autom. Mag.*, vol. 41, no. 6, pp. 593–601, 1994.
[43] G. Tuncel, G. Mirac Bayhan, Applications of Petri nets in production scheduling: a review, *The International Journal of Advanced Manufacturing Technology*, vol. 34, n. 7, pp, 762-773, 2007.
[44] W.M.P. Van Der Aalst, K.M. Van Hee, A.H.M. Ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, M.T. Wynn, Soundness of workflow nets: Classification, decidability, and analysis, Formal Aspects of Computing, 23 (3), pp. 333-363, 2011.
[45] H. Wang, L. Grigore, U. Buy, M. Lehene, H. Darabi, Enforcing Periodic Transition Deadlines in Time Petri Nets With Net Unfoldings, *IEEE Trans. SMCA,* vol. 41, no. 3, pp. 522-539, 2011.
[46] J. Wang, Y. Deng and M. Zhou, Compositional time Petri nets and reduction rules, in *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 30, no. 4, pp. 562-572, 2000.
[47] N. Q. Wu and M. C. Zhou, Schedulability analysis and optimal scheduling of dual-arm cluster tools with residency time constraint and activity time variation, IEEE Transactions on Automation Science and Engineering, vol. 9, no. 1, 203-209, 2012.
[48] N. Q. Wu and M. C. Zhou, Modeling, analysis and control of dual-arm cluster tools with residency time constraint and activity time variation based on Petri nets, IEEE Transactions on Automation Science and Engineering, vol. 9, no. 2, 446-454, 2012.
[49] N. Q. Wu, F. Chu, C. B. Chu, and M. C. Zhou, Petri net modeling and cycle time analysis of dual-arm cluster tools with wafer revisiting, IEEE Transactions on Systems, Man, & Cybernetics: Systems, vol. 43, no. 1, 196-207, 2013.
[50] N. Q. Wu, M. C. Zhou, L. P. Bai, and Z. W. Li, Short-term scheduling of crude oil operations in refinery with high fusion point oil and two transportation pipelines, Enterprise Information Systems, vol. 10, no. 6, 581-610, 2016.
[51] F. J. Yang, N. Q. Wu, Y. Qiao, M. C. Zhou, and Z. W. Li, Scheduling of single-arm cluster tools for an atomic layer deposition process with residency time constraints, IEEE Transactions on Systems, Man, & Cybernetics: Systems, vol. 47, no. 3, 502-516, 2017.
[52] H. H. Xiong and M. Zhou, Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search, *IEEE Trans. Semicond. Manuf.*, vol. 11, no. 3, pp. 384–393, 1998.
[53] G. Xu and Z. M. Wu, "Deadlock-free scheduling strategy for automated production cell," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 1, pp. 113–122, 2004.
[54] E. Yamalidou, J.O. Moody, M.D. Lemmon, P.J. Antsaklis, Feedback control of Petri nets based on place invariants. Automatica, 32(1), 15–28, 1996.
[55] H. Yu, A. Reyes, S. Cang, S. Lloyd,Combined Petri nets modelling and AI-based heuristic hybrid search for flexible manufacturing systems-part II. Heuristic hybrid search. *Comput Ind Eng* 44(4):545–566, 2003.

Appendix

In order to insure the unicity of the structure tree, let us define a total order, referred to as "<<", within the nodes of the tree. This order is used to decide the next step of the construction at each iteration. Consequently, it is also used to index the nodes of *ST*.

Definition 13: The order "<<" is defined within the nodes of the *ST* tree as:

1) $ST\{j\} << ST\{j'\}$ if j is a node obtained at iteration k, j' is a node obtained at iteration k' and $k < k'$,
2) $ST\{j\} << ST\{j'\}$ if j is either a simple transition or a S node and j' is either a C node or a P node
3) $ST\{j\} << ST\{j'\}$ if j, j' are both simple transitions or S nodes (resp. C or P nodes) and $j < j'$.

The first rule gives a priority that depends on the iteration at which the nodes are created. The second rule gives a priority that depends on type of the elementary structure at iteration i: simple transitions t and elementary sequence structures (S) have priority upon elementary choice structures (C) and elementary parallelism structures (P) (then the order within t and S, on one hand and C and P, on the other hand, is decided with the second rule). The third rule gives a priority depending on the indexes of the already existing nodes (the q first nodes are indexed exactly as the transitions). Algorithm 1 details how the ST tree is constructed with the order << from the incidence matrices and the transitions indexes of a given PN. The application of the order "<<" with the design strategy detailed within Proposition 1 leads also to the unicity of the resulting tree:

Proposition 6: The tree *ST* of a given structured net is unique with respect to the indexes of the transitions.

Proof: The demonstration is recurrent. At initialization the nodes of the tree reduce only to simple transitions that are ordered according to their indexes (according to the rule 2 of order <<) and placed in a list $\boldsymbol{E_1}$. Consequently, the initialization of the tree is unique (for some arbitrary indexing of the transitions). Now assume that a partial tree with k nodes has been designed in a unique way up to iteration i. Let us define $\boldsymbol{E_i}$ as the initial list of i-iteration nodes and consider the two following stages (according to the rule 1 of order <<).

At stage 1, the nodes $ST\{j\}$ in $\boldsymbol{E_i}$ are sorted according to their index (i.e. $ST\{j\}.label$). The node in first position (i.e. with the smallest index) is considered and removed from $\boldsymbol{E_i}$. There may exist at most two other nodes in $\boldsymbol{E_i}$ that form a sequences with $ST\{j\}$. In case such nodes exist, the one, namely $ST\{j'\}$, with the best position (i.e. with smallest index) is considered and also removed from $\boldsymbol{E_i}$. $\{ST\{j\}, ST\{j'\}\}$ is an elementary sequence. The next reduction rule replaces this elementary sequence with a new node $ST\{k+1\}$ that is added in last position in list $\boldsymbol{E_i}$. Consequently, the definition and position in $\boldsymbol{E_i}$ of node $ST\{k+1\}$ is unique (for some arbitrary indexing of the transitions). The same reduction rules are applied to $\boldsymbol{E_i}$ as long as elementary sequences can be founded within the nodes of $\boldsymbol{E_i}$.

Let us assume that a series of elementary sequence reductions have been successfully done within the nodes of $\boldsymbol{E_i}$ (k' new nodes have been created) and that no additional elementary sequence reduction is possible within $\boldsymbol{E_i}$. If $\boldsymbol{E_i}$ is composed by a single node then, the construction ends, this node will be the root node of the tree and the construction is unique. Otherwise, at stage 2, the node $ST\{j\}$ in first position in $\boldsymbol{E_i}$ (i.e. with the smallest index) is considered and removed from $\boldsymbol{E_i}$ (note that the nodes in $\boldsymbol{E_i}$ are almost sorted according to their indexes). As long as the net is structured there may exist one or more other nodes in $\boldsymbol{E_i}$ that form elementary choices or parallelisms with $ST\{j\}$. Within these candidates, the node, namely $ST\{j'\}$, with the best position (i.e. with smallest index) is considered and also removed from $\boldsymbol{E_i}$. $\{ST\{j\}, ST\{j'\}\}$ is an elementary structure, no matter the structure is a C or P one, it is uniquely defined according to the node indexes. The next reduction rule replaces this elementary structure with a new node $ST\{k+k'+1\}$ that is added in last position of list $\boldsymbol{E_i}$. Another time, the definition and position in $\boldsymbol{E_i}$ of node $ST\{k+k'+1\}$ is unique. The same reduction rules are repeated to $\boldsymbol{E_i}$ as long as elementary C or P structures can be founded within the nodes of $\boldsymbol{E_i}$. When all nodes in $\boldsymbol{E_i}$ have been explored and no additional structure reductions can be performed, then the list $\boldsymbol{E_{i+1}}$ is initialized with the remaining nodes in $\boldsymbol{E_i}$. If $\boldsymbol{E_{i+1}}$ is composed by a single node then, the construction ends and this node will be the root node of the tree. Otherwise, the construction continues with list $\boldsymbol{E_{i+1}}$, defined in a unique way, at iteration i+1.

A direct corollary of Proposition 6 is that the size of the tree does not depend on the transition indexes in the TPN. It is equal to $2 \times q - 1$. Changing the transition indexes will only lead to a permutation of tree nodes.