Simulation Modelling Practice and Theory 18 (2010) 1225–1241

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Revisiting state space exploration of timed coloured petri net models to optimize manufacturing system's performance

Miguel Mujica *, Miquel Angel Piera, Mercedes Narciso

Autonomous University of Barcelona, Department of Telecommunications and Systems Engineering, 08193 Bellaterra, Barcelona, Spain

ARTICLE INFO

Article history:
Received 18 September 2009
Received in revised form 6 April 2010
Accepted 22 April 2010
Available online 23 May 2010

Keywords:
Coloured petri nets
Timed state space exploration
Simulation-based optimization
Decision support systems
Scheduling
Manufacture
Heuristics

ABSTRACT

Due to constant fluctuations in market demands, nowadays scheduling of flexible manufacturing systems is taking great importance to improve competitiveness. Coloured Petri Nets (CPN) is a high level modelling formalism which have been widely used to model and verify systems, allowing representing not only the system's dynamic behaviour but also the information flow. One approach that focuses in performance optimization of industrial systems is the one that uses the CPN formalism extended with time features (Timed Coloured Petri Nets) and explores all the possible states of the model (state space) looking for states of particular interest under industrial scope. Unfortunately, using the time extension, the state space becomes awkward for most industrial problems, reason why there is a recognized need of approaches that could tackle optimization problems such as the scheduling of manufacturing activities without simplifying any important aspect of the real system. In this paper a timed state space approach for properties verification and systems optimization is presented together with new algorithms in order to get better results when time is used as a cost function for optimizing the makespan of manufacturing systems. A benchmarking example of a job-shop is modelled in CPN formalism to illustrate the improvements that can be achieved with the proposed implementations.

1. Introduction

Nowadays special attention is paid to improving transport and production operations considering efficiency of the whole process. Due to time restrictions in some operations, there is a recognized need of decision support tools to deal with a proper coordination for activities involving logistic operations in a reasonable computational time. Evaluation of several real system scenarios to determine the best decision variable values by means of simulation is an accepted approach due to the ability to support any description level of the system under study. The main drawback of digital simulation as a decision support tool is the difficulty to evaluate more than a reduced number of scenarios for a given system [13]. In order to ensure optimality for the decision taken it is necessary to explore all the possible states a system can reach from a certain system operational context (i.e. initial conditions). However, several problems appear due to computational shortages (memory and time requirements). In this paper an efficient approach to support exhaustive search using the state space to deal with the optimal sequence of events that minimize the makespan is presented.

* Corresponding author. Tel.: +34 935813487; fax: +34 935814031.
E-mail addresses: MiguelAntonio.Mujica@uab.es (M. Mujica), MiquelAngel.Piera@uab.es (M.A. Piera), Mercedes.Narciso@uab.es (M. Narciso).

1569-190X/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.simpat.2010.04.010

1.1. Simulation–optimization approaches

In order to combine the high description capabilities of simulation models with the benefits of analytical techniques of optimization models there have been proposed several simulation–optimization approaches. One of the most classical and widely accepted has been the parameterization of the decision variables of the simulation model in such a way that an optimization algorithm such as evolutionary algorithms can efficiently check the results of the most promising decision variable values. At the end of the procedure it compares the different system outputs and keeps the best of the solutions obtained [6]. This kind of approach has been traditionally implemented in simulations software as optimization packages for most of the available commercial simulators such as OptQuest [17] for ARENA [1] and WITNESS [23] or the SimRunner [21] for PROMODEL [20]. The main advantage of using this approach is that it can support the specification of system dynamics at both macro and micro abstraction level in combination with optimization techniques, requiring only the parameterization of the model in order to get good results. Unfortunately not all the logistic or manufacture problems can be parameterized, in most of them the solution depends highly in the precedence of decisions taken. Thus the evaluation of the different configurations of a scheduling and routing problem with dynamic priorities according to each particular state of the system can not be properly addressed by a parametric approach. Another simulation–optimization approach is the one that allows evaluating all the possible system configurations with the analysis of different alternatives exploring the state space of the simulation model. The advantage of using this approach is that since theoretically it is possible to evaluate all the possible configurations of a system the obtained solution is also the optimal one, a result which can not be ensured with other simulation–optimization approaches. The main disadvantage of using this state space approach is that commonly it appears the state explosion problem [22] when relatively small models have a quantity of states so big that results unpractical to explore all of them or sometimes impossible due to resource computer limitations. In this paper an alternative to alleviate as much as possible this state explosion problem based on the analysis of state space of Coloured Petri Net models will be presented. Coloured Petri Nets (CPN) is a modelling formalism which allows modelling properly the dynamic behaviour of systems at different abstraction levels. CPN fully supports the state space analysis in such a way that optimization problems can be solved as search problems. There are several CPN academic modelling software's which implement the state spaces analysis such as CPNTools [3] or PetriSimM [18] but the implemented state space tools are mainly oriented to qualitative analysis to evaluate model properties such as deadlocks, liveness, boundedness among others.

1.2. Timed coloured petri nets

Coloured Petri Nets (CPN) is a simple yet powerful modelling formalism, which allows the modelling of complex systems which present an asynchronous, parallel or concurrent behaviour and can be considered discrete event dynamic systems [9,14]. The formalism allows modelling not only the dynamic behaviour of systems but also the information flow, which is an important characteristic and very useful in industrial systems modelling.

In order to investigate the KPI's (Key Performance Indicators) at which the industrial systems operate under different policies, such as scheduling, resource occupancy, costs and inventory among others it is convenient to extend CPN with a time concept (TCPN). This extension is made introducing a global clock for the model and time stamps for the entities. The global clock represents the model time, and the time stamps describe the earliest model time at which the entities of the model, graphically represented by dots (tokens), can be used for the transition evaluation process [10]. A token is *ready* if the correspondent time stamp is less than or equal to the current model time. If the token is not ready, it can not be used in the transition enabling procedure.

The formalism can be graphically represented by a bipartite graph where the place nodes are represented by circles and the transition nodes by rectangles or solid lines. Fig. 1 helps illustrating the definition of the TCPN.

The model consists of a finite set of place nodes P, a finite set of transition nodes T and a finite set of directed arcs A. For the example given in Fig. 1, P, T and A are defined as follows:

$$P = \{P_1, P_2, P_3\},$$
$$T = \{T_1, T_2, T_3\},$$
$$A = \{(P_1, T_1), (P_1, T_2), (T_1, P_2), (T_2, P_2), (P_2, T_3), (T_3, P_3)\}.$$

The colour sets Σ which are used in the example of Fig. 1 are declared in Table 1.

The set of free variables V are the ones that take values over the correspondent colour set where they participate, in the case of the example of Fig. 1 the variables used are defined in Table 2.

The function $C : P \to \sum$ assigns to each place p a colour set belonging to the set of types Σ. For the example of Fig. 1 it is defined as:

$$C(P) = \begin{cases} \text{Information} & \text{if } p = P_1, \\ \text{NO} \times \text{Value} & \text{if } p \in \{P_2, P_3\}. \end{cases}$$

The *guard function* assigns to each transition t a guard $G(t)$ which is an expression whose evaluation is required to be a Boolean expression i.e. Type [$G(t)$] = BOOLEAN. The set of variables that appear in the guard expression is required to form a subset of V. The guard function for example of Fig. 1 is defined as:

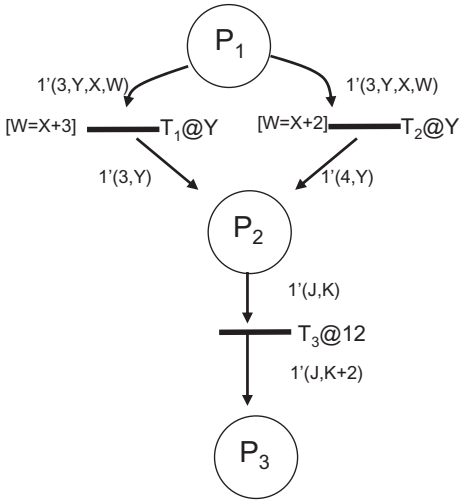


Fig. 1. Example used to illustrate the formal definition.

Table 1
Colour set definition.

Colour sets	
NO	NO = {3,4}
VALUE	VALUE = integer
NO × VALUE	Product NO × VALUE
INFORMATION	Product NO × VALUE × VALUE × VALUE

Table 2
The variable set.

Variables
X, Y, W, K: VALUE
J: NO

$$G(t) = \begin{cases} W = X + 3 & \text{if } t = T_1, \\ W = X + 2 & \text{if } t = T_2, \\ \text{TRUE} & \text{otherwise.} \end{cases}$$

The arc expression function $E(a)$ assigns to each arc a an arc expression $E(a)$. In this case the variables that participate in the expressions are required to be a subset of V. For the example of Fig. 1 the arc function is defined as follows:

$$E(a) = \begin{cases} 1'(3Y, X, W) & \text{if } a \in \{(P_1, T_1), (P_2, T_2)\}, \\ 1'(3Y) & \text{if } a = (T_1, P_2), \\ 1'(4Y) & \text{if } a = (T_1, P_2), \\ 1'(J, K) & \text{if } a = (P_2, T_3), \\ 1'(J, K + 2) & \text{if } a = (T_3, P_3). \end{cases}$$

Using this formalism for the modelling and evaluation of manufacturing systems, it is natural to associate activities of the real system to transitions in the CPN model. Simulation community use TCPN to specify discrete event systems by attaching a Δr time delay to transitions in order to simulate time consumption of a certain activity. Therefore when a transition occurs, the output tokens will have a time stamp Δr time units larger than the current global clock Gc which simulates time delay due to the execution of an activity. The time stamp calculated by Eq. 1 represents the earliest model time when the output tokens can be used again for a new transition firing, i.e. the token will not be available for Δr time units. This characteristic allows modelling time consumption of a real system activity.

$$t_o = Gc + \Delta r, \tag{1}$$

where t_o is the time stamp value that must be attached to the output tokens when the transition firing takes place, Gc is the global clock of the model when the firing occurs and Δr is the time associated with the transition.

It is a common convention to use the sign @ to denote time in the elements of the model. When it is attached to transitions, it specifies the time consumption Δr. In the example of Fig. 1 the function $D(t)$ assigns the consumption time for each transition t in the model:

$$D(t) = \begin{cases} Y & \text{if } t \in \{T_1, T_2\}, \\ 12 & \text{if } t = T_3. \end{cases}$$

The initialization function $I(p)$ assigns to each place p an initialization expression which is required to evaluate over the colour set of the place p. One initialization function for the example in Fig. 1 (not depicted in the figure) could be:

$$I(p) = \begin{cases} 1'(3,5,3,6)@[0] + 2'(3,4,5,3,5)@[0,0] & \text{if } p = P_1, \\ 0 & \text{otherwise}. \end{cases}$$

The formal definition of TCPN is the following one.

Definition 1. The non-hierarchical TCPN is the tuple:

$$\text{TCPN} = \left(P, T, A, \sum, V, C, G, E, I\right), \text{ where}$$

1. P is a finite set of places.
2. T is a finite set of transitions T such that $P \cap T = \emptyset$
3. $A \subseteq P \times T \cup T \times P$ is a set of directed arcs
4. $\sum$ is a finite set of non-empty colour sets.
5. V is a finite set of typed variables such that Type $[v] \in \sum$ for all variables $v \in V$.
6. $C : P \to \sum$ is a colour set function assigning a colour set to each place.
7. G: $T \to$ EXPR is a guard function assigning a guard to each transition T such that Type $[G(T)]$ = Boolean.
8. E: $A \to$ EXPR is an arc expression function assigning an arc expression to each arc a, such that:
   Type $[E(a)] = C(p)$
   Where p is the place connected to the arc a
9. I is an initialization function assigning an initial timed marking to each place p such that:
   Type $[I(p)] = C(p)$

EXPR denotes the mathematical expressions associated to the elements of the formalism (variables, colours, logic conditions) where the syntax can vary when coding the formalism in a programming language. The TYPE $[e]$ denotes the type of an expression $e \in$ EXPR, i.e. the type of values obtained when evaluating e. The set of free variables in an expression e is denoted VAR $[e]$ and the type of a variable v is denoted TYPE $[v]$.

The *state* of every CPN model is also called the *marking* which is composed by the expressions associated to each place p and they must be closed expressions i.e. they can not have any free variables. On the other hand in the case of TCPN the timed markings are composed by these expressions together with their time stamps. Formally they can be defined as follows:

Definition 2. The *timed marking* of a TCPN is a function $M : P \to$ EXPR such that $M(p) \in C(p)$. It maps each place p into a multi set of values $M(p)$ representing the timed marking of place p. The individual elements of the multi set are called *timed tokens* and the expressions contain also the time information (time stamps).

Definition 3. The *untimed marking* M_U of a TCPN model is a function $M_U : P \to$ EXPR that maps each place p into a multi set of values $M_U(p)$ representing the untimed marking of place p and $M_U(p) \in C(p)$. In this case the expressions do not contain any time information.

Definition 4. The *objective marking* is a particular configuration of tokens in places disregarding the time extension, i.e. a particular untimed marking M_U.

1.3. Timed state space analysis

The reachability graph is a directed graph used commonly for the verification and analysis of behavioural properties of CPN models. This analysis is done through the generation and storage of all the different reachable states from an initial one, reason why it is also known as the State Space (SS). The reachability graph has been used with timed or untimed models to evaluate properties, such as liveness, boundedness and reachability of states among others [2,11], to determine the behaviour of the modelled system.

In the SS each node represents a timed marking of the TCPN model. The following definitions are used through the rest of the article for the analysis of the timed coloured Petri net models.

Definition 5. A state M_k will be considered as *old node* if it is exactly the same (together with its time values) as one M_j that had been previously generated in any other level of the SS, i.e. $M_k = M_j$.

Definition 6. A *dead marking* is a state that does not have any enabled transition.

Definition 7. A *new state* is a node that is neither a dead marking nor an old node.

Definition 8. A *feasible path* is a sequence of nodes that go from the root node or initial state to the objective marking.

The main characteristics of the SS are:

- The root node represents the initial marking of the system.
- The successor or children nodes correspond to the new states or markings obtained once the enabled transitions have been fired.
- For each node in the tree, as many successor nodes must be generated as many enabling combination of tokens the marking has.
- Each node is connected with its successor nodes through directed arcs.
- The connecting arcs represent transition firings and they also contain the information regarding the fired transition and the used tokens.

Fig. 2 illustrates information given in the SS, where node 1 represents the initial marking at 0 time of the TCPN model (left hand side of the figure).

Each node on the right hand side of Fig. 2 represents a timed marking with its own global clock and time stamps of the tokens (not depicted in the figure). The states in the graph are connected with successor states through directed arcs, and the initial state can always be reached if the arcs are traversed in reverse direction e.g. state 10 can be reached from 1 through the connecting arcs that connect nodes 2, 6 and 8. Each arc represents a transition firing with a particular token combination stated by the arc inscription (the @ sign is used to denote time) e.g. from state 1 to state 2 the inscription $T_2@0 : 1\prime(3)@0, 1\prime(1)@0$ means that transition T_2 is fired at global clock 0 using tokens 1'(3) from place node P_1 with time stamp 0 and 1'(1) from place node P_5 with time stamp 0. The time stamps for each new state must be calculated using formula1.

1.3.1. Event driven SS vs. time driven SS

When dealing with timed state space, some authors use a kind of *Time Driven State Space* (TDSS) or *Event Driven State Space* (EDSS) as basis for the generation of successor states. The difference between both approaches lies in the fact that for a specific global time, the TDSS considers only in the SS the transitions that are enabled at the same global time, and generates as many successors as enabled transitions the marking has. When all the possible transitions have been fired, the global clock heads to the next time value at which at least one transition is enabled among all the states [10].

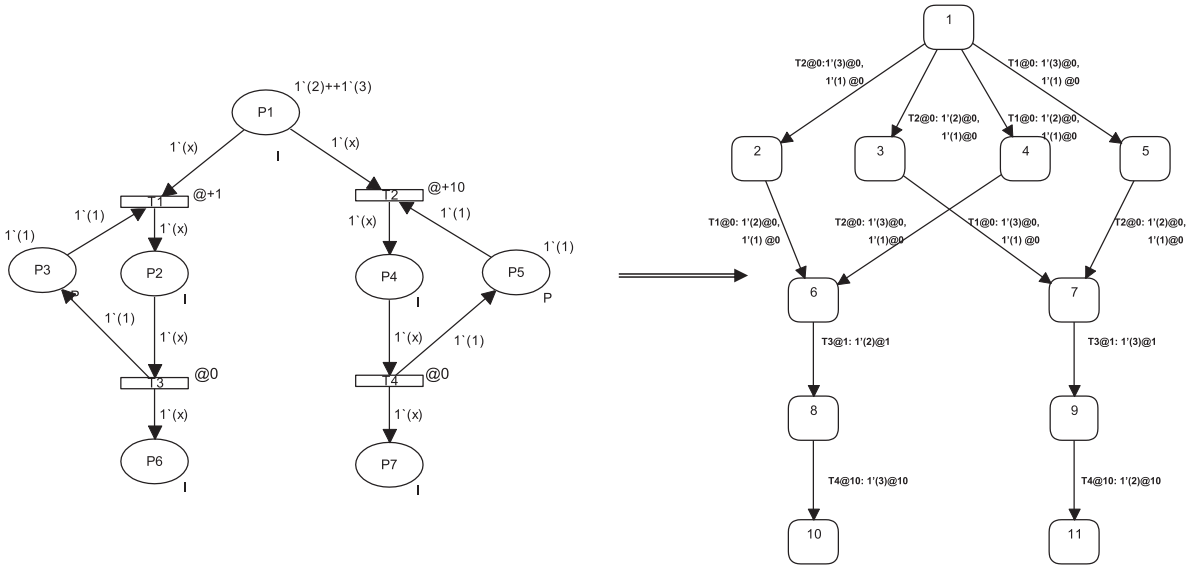


Fig. 2. The state space.

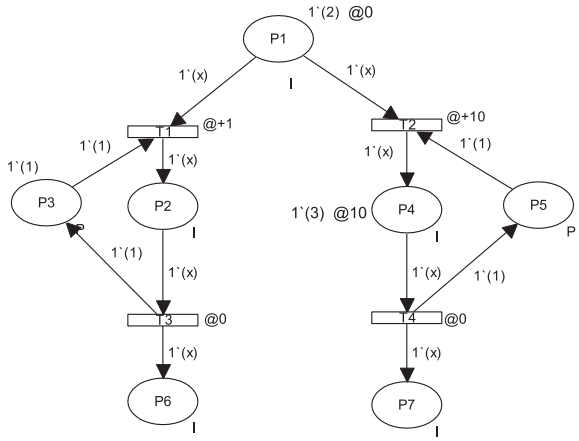


Fig. 3. CPN model of node 4 in the TDSS or node 5 in the EDSS with transitions in conflict.

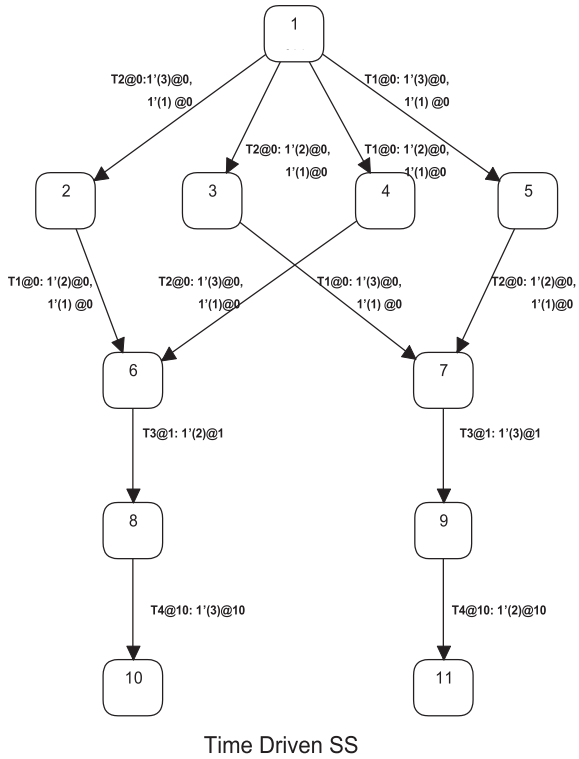


Fig. 4. Time driven state space.

On the other hand, the EDSS generates the successors of a state firing the transitions that are enabled without taking into account time restrictions (event). The latter is resolved advancing the global clock of a state to each earliest time at which transitions can be fired. Using this approach it includes all the possible events or fireable transitions from the same marking resulting a SS wider (in the number of states sense) than the TDSS e.g. if two transitions T_1 and T_2 are in conflict and T_1 is enabled before T_2, TDSS will generate only the successor from the firing of T_1 but with the EDSS approach it generates the resulting successor state from the firing of T_1 and also the successor from the firing of T_2.

In order to investigate industrial models performance, a proper timed approach must be chosen to evaluate all the possible configurations of a system without loosing optimal configurations that could occur when the state space generation takes place [19].

Figs. 4 and 5 illustrates these two approaches. On the one hand with the TDSS mechanism (Fig. 4), the total number of evaluated states is less than the one obtained with the EDSS (Fig. 5).With the TDSS approach the states from the second and third level generate only one new state due to the conflicting transitions, while with the use of EDSS allows evaluating

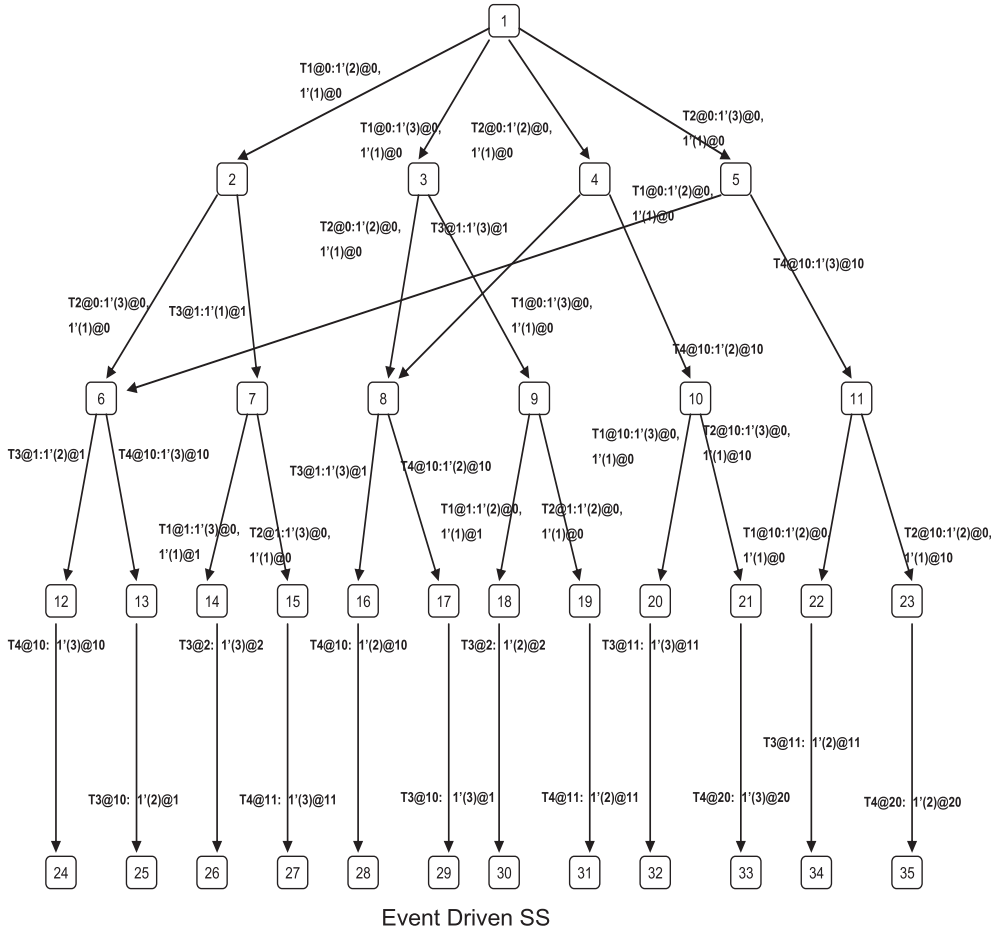


Fig. 5. Event driven state space.

the two possible states (a situation that definitively can happen in the real system). To exemplify this situation let us check node 4 in the TDSS (Fig. 4) that corresponds to node 5 of the EDSS (Fig. 5). The corresponding CPN model for both approaches is the one presented in Fig. 3. Node 4 in Fig. 4 is the result of firing transition T_2 at 0 global time using tokens 1'(3) from place P_1 and 1'(1) from place P_5. Since the TDSS is time oriented, the only transition that can be fired is transition T_1 (at 0 global time) reason why node 4 of the TDSS has only one successor.

The TDSS approach allows taking advantage of the time policy to implement techniques such as the *sweep line method* [2,12] which has been proved to give good results for verification and validation purposes.

On the other hand, due to the fact that EDSS (Fig. 5) considers for each state (a node in the SS) all the transitions enabled by colour without considering if tokens are ready (event oriented), it can support the evaluation of different system configurations that definitely occur in real flexible manufacturing systems and that can not be generated with the TDSS approach. As an example, let us analyze again the model presented in Fig. 3 but under an event oriented scope. In this case node 5 of Fig. 5 will generate two different states (nodes 6 and 11). These states are generated taking into account the fact that transition T_1 can be fired at 0 global time, and transition T_4 can also be fired (since it is colour enabled) but the global clock must be advanced to the moment of firing, which in this case is 10 global time (global clock is increased to the minimum time stamp that enables the transition).

Both, TDSS and EDSS can be used to search for a sequence of events that drives the system to a goal state, but applying the TDSS in manufacture models could result in the missing of possible configurations as it has been shown in [19]. By analyzing the complete SS, all the paths (i.e. sequence of events) that deal with feasible solutions (i.e. objective states) can be obtained and consequently can be used by decision makers to improve real system performance [5].

2. An efficient approach for event driven timed state space

State explosion problems appear quite frequently when EDSS is applied to analyze industrial and logistic systems because of inherent flexible mechanisms designed to cope with demand/production perturbations. The increased flexibility leads to a high number of possible operation scenarios which are described mathematically by a huge domain of decision variables.

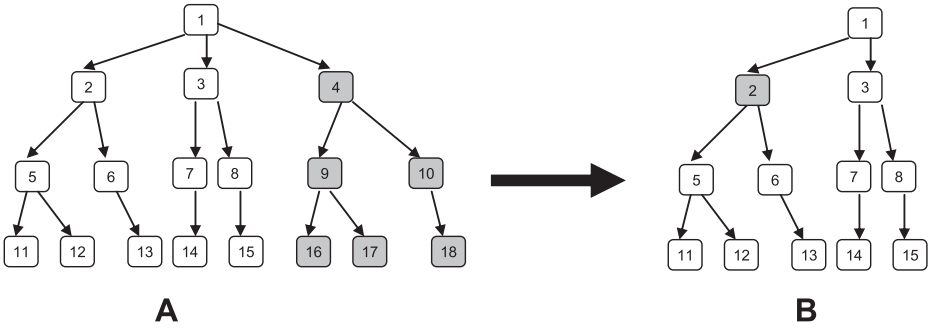


Fig. 6. Use of symmetric old nodes.

Furthermore, when the time extension is added to untimed models with a finite SS, the resulting SS becomes greater than the original one or even infinite [11] causing in most of the cases CPU memory overloading or long-time response in computer tools based on SS analysis. This motivates the development of analysis techniques that can provide an adequate analysis support to deal with these problems [2,7,8].

2.1. Symmetric old nodes

Two states are symmetric when they are distinguishable from each other only by their time stamps. Based on this characteristic the *symmetric old node* (S-old node) is the timed state (node in the SS) whose underlying untimed state is the same as the untimed state of one already generated state. When dealing with S-old nodes, the successor nodes that hang from an S-old node (sub tree) will be also symmetric to the ones that hang from the original state. Next figure exemplifies the concept of old node symmetry.

In Fig. 6(A), node 4 represents an S-old node of node 2, in this case all the states in the sub tree of 4 (nodes 9, 10, 16, 17 and 18) will be S-old nodes of the sub tree of state 2 (nodes 2, 5, 6, 11, 12 and 13) since the two sub trees differ from each other by their time stamps.

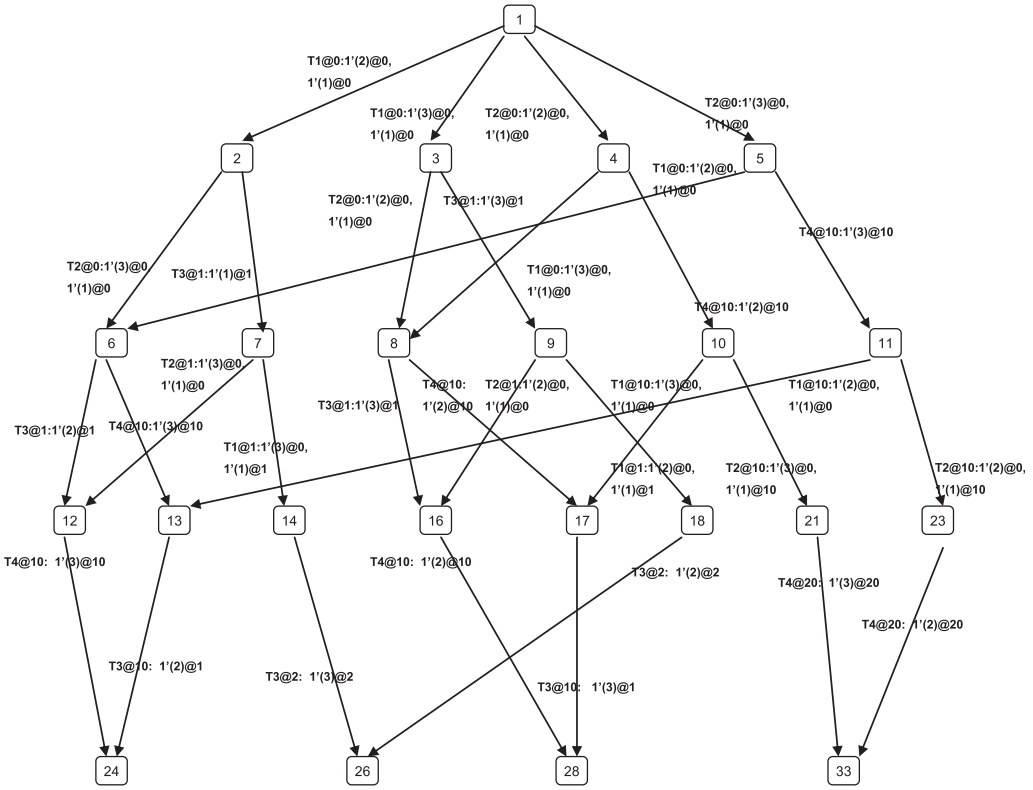


Fig. 7. Compacted EDSS.

Using S-old nodes it is possible to reduce the amount of memory resources to store all the SS information. This can be achieved using the S-old nodes to take advantage of the symmetries avoiding storing the sub trees of the repeated S-old nodes. The latter is achieved storing only the root of the different S-old nodes together with the time values of the repeated S-old nodes for later analysis. The feasible reduction is exemplified in Fig. 6(B), where node 2 represents the S-old node stored with a reference of the time values of the other S-old node (node 4). If the time information of the S-old node 4 and the sub tree that hangs from it is needed, it can be generated using the information stored along with the S-old node 2 and the sub tree that hangs from node 2.

The S-old nodes represent in manufacture or logistic systems the same logistic state but reached at different times (the time values of the different S-old nodes).

2.2. Compact representation of a timed EDSS

When dealing with timed SS of manufacture models, it is common to find a considerable amount of symmetries in the underlying untimed state space. Therefore, it is possible to take advantage of the S-old nodes to represent and store the state space maintaining the necessary information for the state space analysis. If the S-old nodes are stored only once and the different timed values stored apart, the resulting tree is smaller than the original. It is remarkable that this approach would be very useful also in stochastic nets where time is not deterministic since with the use of typical SS approaches the resulting SS would result intractable not only because the state explosion but also because the time differences between states. Fig. 7 exemplifies the resulting SS if the SS of Fig. 5 is represented using the S-old nodes instead of the old nodes.

It can be seen from Figs. 5 and 7 that using S-old nodes reduce the total number of states from 35 to 23 states (the original state number was left on purpose in Fig. 7 to appreciate the S-old nodes). It can also be appreciated that the size reduction increases the amount of old nodes to analyze (from 2 old nodes to 10 S-old nodes). Table 3 shows the markings that correspond to the S-old nodes in the Compacted EDSS (CEDSS) of Fig. 7.

Using this approach there is a considerable reduction in SS size, but more analysis efforts must be made in order to evaluate the sub tree information through the S-old node analysis. The SS analysis approach presented in the following sections will use the CEDSS to analyze systems performance; in order to give a better description of the evaluation mechanism the node notation will use sub indexes from this section on.

2.3. The S-old node analysis approach

In [16] an approach that reduces the state explosion problem avoiding the exploration of some SS branches by analyzing the time stamp values of S-old nodes is presented. As an example let us consider Fig. 8; it presents an M_J state which is the S-old node of the previously generated M_i state. In this case since the time values of M_J will produce greater time values than the ones of successors of M_i, and the objective is minimizing the time values of the objective state, it is not convenient to explore the branches that can be generated from the M_J state. Based on the fact that the underlying untimed successors will be the same, it is meaningless to compute the tokens and the attached colours of successor nodes of a repeated S-old node.

Given any two S-old nodes M_i (the first generated state) and M_J, and the time stamp lists of size k of each node T_i and T_J, let us explain the decision procedure by defining the Ψ and Φ_k functions as follows:

$$\Psi: \quad \mathbb{R}\times\mathbb{R}\to\{0,1\},\qquad (x,y)\mapsto\begin{cases}0, & x\geqslant y,\\ 1, & x<y,\end{cases} \tag{2}$$

$$\Phi_k: \quad \mathbb{N}^k\times\mathbb{N}^k\to\mathbb{N},\qquad (T_i,T_j)\mapsto\sum_{l=1}^{k}\Psi(T_{il},T_{jl}). \tag{3}$$

Table 3
The resulting S-old nodes.

Node	Coloured marking
6	[0,1(2),0,1′(3),0,0,0]
8	[0,1(3),0,1′(2),0,0,0]
12	[0,0,1′(1),1′(3),0,1′(2),0]
13	[0,1(2),0,0,1′(1),0,1′(3)]
16	[0,0,1′(1),1′(2),0,1′(3),0]
17	[0,1(3),0,0,1′(1),0,1′(2)]
24	[0,0,1′(1),0,1′(1),1′(2),1′(3)]
26	[0,0,1′(1),0,1(1),1′(2) + 1′(3),0]
28	[0,0,1′(1),0,1′(1),1′(3),1′(2)]
33	[0,0,1′(1),0,1′(1),0,1′(2) + 1′(3)]

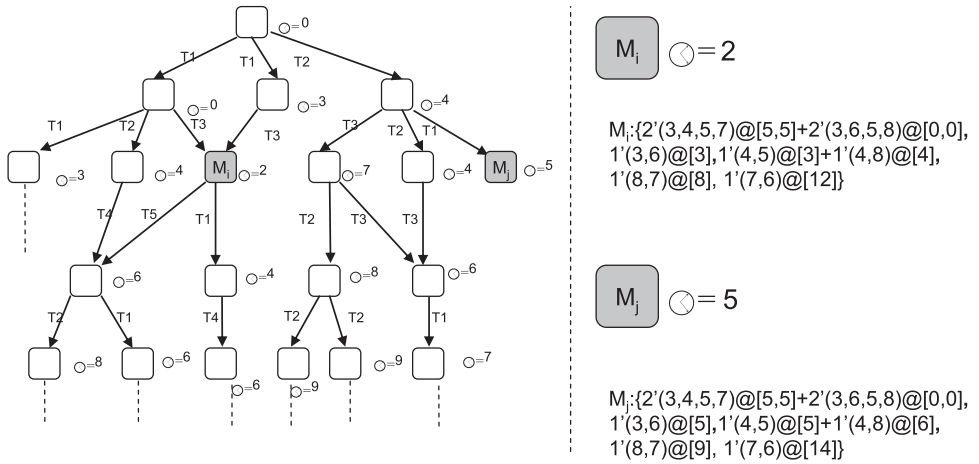


Fig. 8. Time analysis of the repeated states.

Applying the function Φ_k to the S-old nodes M_i and M_J (M_i is the S-old node that appeared first in the SS and M_J the S-old node with different time stamps) the following three possible outcomes can be obtained:

(a) $\Phi_k = 0$: The time stamp values of the M_i state are equal or higher than the ones of the M_J state. In this case the best branch is chosen from among the successors of M_i (since the best branch of M_i will also be the best one for M_J) and the time values of the path that goes to the objective state are updated by analytical evaluation of the new time values using the time stamps of the M_J state. The previous procedure is done using token and transition information stored from the generation of M_i.

(b) $\Phi_k = k$: The time stamp values of the M_i state are smaller than the ones of M_J state. In this case, it is not necessary to evaluate again the successor nodes of the M_J state because the underlying untimed states will be the same as those of M_i, and the time values will not be smaller than the ones generated by M_i. The previous procedure is done using token and transition information stored from the generation of M_i.

(c) $0 < \Phi_k < k$: Some time stamp values of the M_J state are equal or higher than the ones of M_i state and others are less than the ones of M_i state. In this case it is not possible to decide whether the M_J state produces better results in the objective marking or not. Therefore an exploration through the best branch has to be done until the objective marking is found and then the final comparison can be made.

This decision procedure gives good results in academic implementations such as the 3×3, 5×5, 6×6 job-shops among others [15,16]. This approach can also be used altogether with the CEDSS to develop an algorithm that minimizes as much as possible the state explosion problem. In the following sections some implementations are proposed in order to improve its computational efficiency to tackle the CEDSS of real systems.

2.4. Verifying and optimizing industrial systems in two steps

A two-step approach is proposed for the generation and analysis of CEDSS models. The first step consists in the generation of all the different states of the model. The second step performs the analysis of the S-old nodes. This approach has the advantages that if the user wants to verify some behaviour properties of the model (such as deadlocks, reachable states, etc.) it can be done in the first step, and the process can be stopped at the end of this procedure. Nevertheless the S-old nodes can be analyzed using a cost function in the second step if the user wants to perform an optimization based on a specific time criterion.

2.4.1. State generation phase

In the first step, a CEDSS is generated. The algorithm generates as many new states as different token combinations enable the transitions based on the restrictions imposed by the colours. The firing of transitions happens immediately, the new states are generated and the correspondent time values of the new states are calculated so that the global clock synchronizes with the earliest firing time.

The successor states are generated using a *Depth-First Search* (DFS) algorithm. For this purpose two lists are necessary for the implementation of the two-step algorithm. One list (SS list) is used to store all the different states with their time elements (time stamps and global clock) and the information of the tree structure (parent–children relationship). A second list (ON-list) is used for the storing of S-old nodes; it stores only once the structure information (tokens and colours) of the S-old nodes that are different from each other and the time values of the repeated S-old nodes found during the SS generation.

While the SS structure is stored using the SS list, the optimization task is performed using the ON-list at a second process. During the first step, each time an S-old node is found, it is not evaluated again (based on the fact that the underlying untimed successors are the same), but its time elements are stored in the ON-list with reference to the correspondent repeated S-old node that had been previously generated in the SS. Due to the characteristics of the S-old nodes, the successor nodes from a repeated S-old node will be considered the same as those generated from the original state, however their time values will be different. Therefore the sub tree of the repeated S-old nodes will be considered the same of the one S-old node that first appeared in the tree. This fact is useful to avoid the storage of information related to the sub trees of the repeated S-old nodes but it requires a second process to analyze the different feasible paths in those nodes of the SS that can deal with the optimal solution.

Fig. 9 illustrates the use of the SS list to store the CEDSS (left hand side) and the ON-list for the S-old nodes (right hand side). On the left hand side of the figure, the gray-shaded M_4, M_5, M_{12} and M_{15}nodes represent the S-old nodes. The right hand side of the figure represents the elements in the ON-list that will be composed by the information of the S-old node in the SS and the time information of the repeated S-old nodes (referenced to the nodes that generate the S-old node). To illustrate this functional concept, let us take as example the S-old node M_4 which can be reached from M_1 and M_3. In this case the correspondent element in the ON-list is composed by the information of the M_4 node and the time information generated by the firings of states M_1 and M_3; in a similar way, the stored information for M_5 is the one generated by nodes M_2 and M_3. The construction of the remaining nodes M_{12} and M_{15} is straightforward from the figure. This management of old nodes allows storing at most the same number of states as the ones for the underlying untimed SS since the S-old nodes are stored and generated once during the first step. This fact does not imply that the occurrences in the untimed and the timed SS are the same since the timed restrictions impose different occurrences of the model.

2.4.2. Time consistency problems in the generation phase

Using a CDESS approach it is important to check that the paths are *time consistent* in the sense that the global clock advances from node to node exactly the time that is consumed by the fired transition, i.e. the successor states are those that are generated using the time values of the parent state stored in the SS list. All the generated states in the first phase are time consistent from the initial node to the first S-old node found for each evaluated path. Once an S-old node is detected, the successors are not calculated again but the time information is stored in the ON-list for the correspondent S-old node so that the information needed for evaluating the different paths is available in the ON-list. Fig. 10 illustrates the time consistency of the different paths in the generation phase.

In Fig. 10 the shaded nodes represent the S-old nodes of a CPN model and the dotted line represent the feasible path generated at the end of the first step. The feasible path that goes from the initial state (node M_1) to the objective one (node M_{15}) is time consistent as well as the paths $[M_1,M_2,M_5,M_9,M_{14}]$, $[M_1,M_4,M_7,M_{12}]$, and $[M_1,M_4,M_8,M_{13},M_{16}]$. The path $[M_1,M_2,M_5,M_{10},M_{15}]$ and the path $[M_1,M_4,M_7,M_{11},M_{15}]$ that reach the objective state are not time consistent since the S-old nodes M_{10} and M_{15} belong to the time consistent path $[M_1,M_3,M_6,M_{10},M_{15}]$. In this example, the M_{10} node colour structure will be stored once and the time elements of the S-old node generated from nodes M_5 and M_6 will be stored in the ON-list.

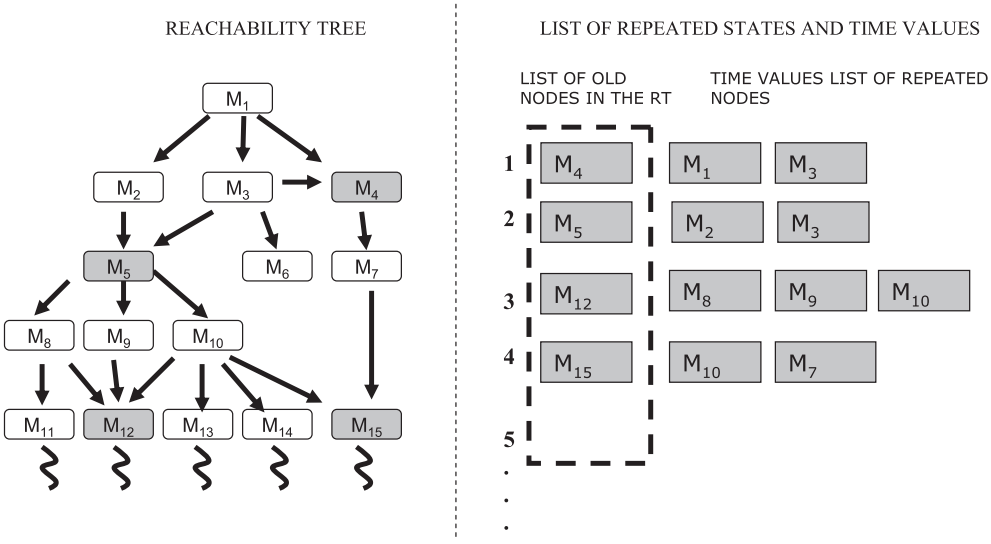


Fig. 9. Information management in the two-step algorithm.

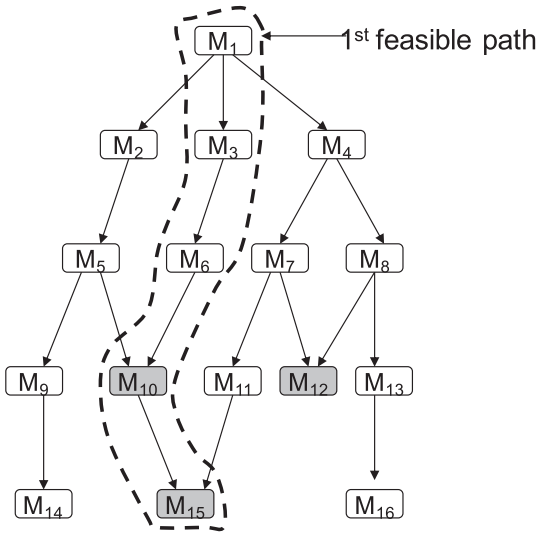


Fig. 10. The time consistent paths.

2.4.3. Second step: optimization phase

In the first step, the DFS algorithm generates a feasible path that goes from the initial to the goal state. It also generates the different paths that go from the initial state to the goal state which are analyzed in the second step with the time information of the S-old nodes stored in the ON-list.

The second step is used for optimization purposes through the analysis of the ON-list. The time analysis of the S-old nodes is done following a simple order in the ON-list (from the first to the last element). It takes an S-old node from the ON-list and compares the time stamp values of the S-old node used in the generation step (left column in Fig. 9) with the time stamp values of its repeated S-old node states using the time analysis presented in Section 2.3. When it detects the time values that are better for the purpose of minimizing the time values, it updates the feasible path with the new time values always maintaining time consistency in the updated path. The analysis determines the S-old nodes that have more chances to deal with an objective state with minimum time.

When this procedure is applied to manufacture or logistic models it optimizes the makespan of the whole procedure.

To exemplify the selection procedure that takes place in the second step, let us consider Fig. 11; it illustrates how the feasible path is updated once the S-old nodes are analyzed at the second step.

The M_4 time values (first element in the ON-list) must be compared to the time values of the repeated S-old state that would be reached from M_3 to check if it produces better time values than those of M_1 (Fig. 11(A)). In the case that M_3 produce better time values than M_1, the feasible path that goes to the objective state is updated with the time information generated from node M_3 (Fig. 11(B)). The same analysis must be done for the rest of the elements in the ON-list (M_5–M_{15}). During this process the time values of the different paths in the SS are updated with the best time values maintaining always the time consistency of the updated paths that lead to the objective state improving progressively the feasible paths. Thus, time consistency is preserved only in those states that can drive the system to the optimal solutions.

This approach presents the following advantages over the approach presented in [16] and over other methods based on exhaustive search:

- The first step can be used only for property verification analysis and for SS analysis of untimed models.
- The second step (used in this work for timed optimization) can be implemented using cost functions such as production cost, throughput, etc.
- With such an approach on industrial problems it is possible to verify some behaviour properties prior to any optimization effort.
- The SS is generated in such a way that it minimizes memory resources overload since it does not store all the information needed in exhaustive search approaches. It also stores the necessary information for analyzing the occurrence paths and preserves time consistency in the best solution found.

2.5. Heuristics to improve the state generation phase

Since optimality is a task achieved by analyzing the S-old nodes, it is expected that the performance evaluation could be improved in the second step varying the sequence of analyzed states by the DFS algorithm (first step). Therefore it is very important to develop a utility function f_k (time function, cost function, performance function) that assigns a value to each

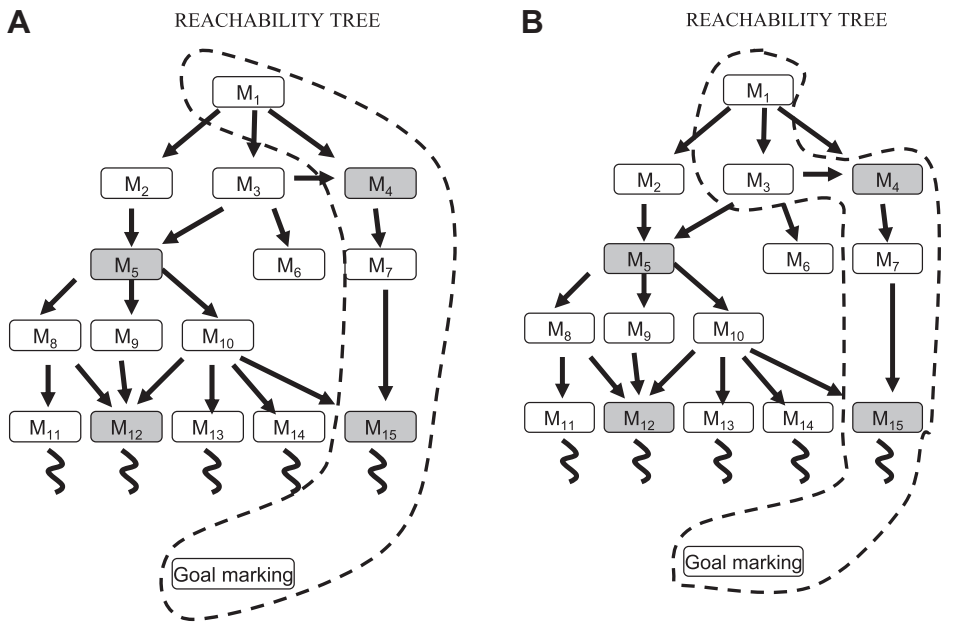


Fig. 11. Updating the feasible path through the second step.

state. Formula (4) can be used by the algorithm in the first step to select the best successor for the next evaluation among the possible successor states.

$$f_k : \mathbb{N}^k \to \mathbb{N}. \tag{4}$$

In the case that no utility function is implemented, the successors would be evaluated without any preference. Using a utility function, the successor to be evaluated is the one with the lowest utility value (minimizing criterion). Fig. 12 illustrates the implementation of the utility function to the selection of successor states. In this example node 4 will be the next state to be evaluated.

Two heuristics are proposed in order to improve the node selection used in the DFS algorithm. This selection allows improving the overall performance of the whole algorithm since it reduces the quantity of deep explorations, which are performed in the second phase. Both heuristics assign a value to the successors of a node. The difference between the two heuristics lies in the selection criterion of the successor nodes; one is based on absolute time stamp values while the other is based on probabilistic assumptions.

During the DFS algorithm, the selection of a node among successors is done based on the value calculated with formula (4). Formula (5) states that the node with the lowest value is the one to be selected for the next evaluation. More formally, given a set of N successors, the next node to be evaluated is selected under the following criterion:

$$\min\{f_k(M_j)\}_{j=1}^{N}. \tag{5}$$

2.5.1. Heuristic based on absolute time values

Since it is not known beforehand which transitions will be fired, the heuristic leads to the evaluations of states based on the assumption that states with the smallest time stamp values have the smallest chances to increment the global clock when subsequent evaluations take place. Function (6) is implemented to give the sum of the time stamps of each marking. Formally, given the T_i time stamps of any state from a set of successors, let F_k be the function that sums the time stamps where the index j goes from token 1 to token k of the marking.

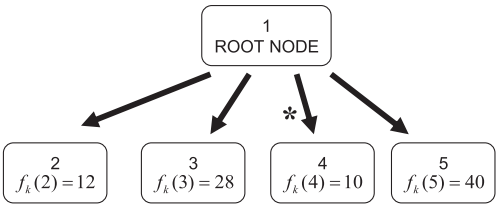


Fig. 12. Utility function for selecting the best node.

$$F_k: \quad \mathbb{N}^k \to \mathbb{N}, \quad T_i \mapsto \sum_{j=1}^{k} T_{ij}. \tag{6}$$

2.5.2. Heuristic based on probabilistic values

The main idea is to compare the time stamp information with the global clock of the state, evaluate how many tokens are greater than the global clock, and how many are smaller. Based on that quantity, formula (7) calculates the probability of increasing the global time (i.e. simulation clock) in the successor state.

Given an M_j state of the reachability tree, let P_k be the function that assigns the probability of increasing the global clock Gc when evaluating the time stamps T_i from token 1 to k in the marking.

$$P_k: \quad \mathbb{N}^k \times \mathbb{N} \to \mathbb{N}, \quad (T_i, Gc) \mapsto \frac{\sum_{j=1}^{k} \Psi(T_{ij}, Gc)}{k}. \tag{7}$$

Using formula (2) the function Ψ assigns value 1 to each time stamp that is greater than the global clock and with (7) the probability is calculated based on the number of tokens that have greater values than the global clock.

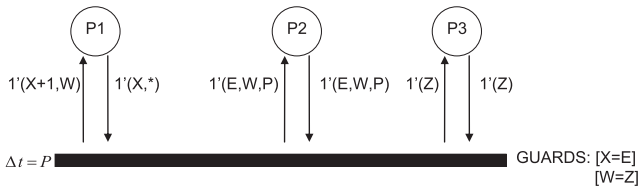
3. The job-shop problem

The two-step approach was coded and it was tested modelling Job-Shop (J-S) problems, which are considered well-known benchmarks [4], and they result useful as test-beds for the algorithm efficiency. The 3×3, 5×5 and 6×6 job-shops have been developed using the TCPN formalism with the purpose to obtain an optimized schedule. Fig. 13 gives an example of the CPN models developed for these problems.

The place node P_1 uses two colours, one is the information of what task is being executed in a certain period of time, and the second colour holds the information about the sequence when the current task ends e.g. one token 1'(11,1) means that the job 1 is executing task 1 (colour 11), and machine number 1 must be needed for the next task (colour 1). Place node P_2 contains the information about the job and sequence of each task (E colour), the machine used for each different task (W colour) and the time consumed when using a particular machine (P colour) e.g. token 1'(22,1,6) means that job 2 in the second task (colour 22) must use machine number 1 (colour 1) and this task will consume 6 time units (colour 6). Place P_3 includes the information about the machine availability during the manufacturing process (Z colour). The initial state of the model is presented in Table 4.

The production target is described in Table 5. It specifies the state where all the jobs have been processed. In this case when colour W equals 8 for each token in place P_1 it determines that the correspondent job is finished.

The key performance indicators of the algorithm are:



COLOUR	DEFINITION	DESCRIPTION
W	Int 1..3	Machine Needed for the Next Job
X	Int 11..33	Job and Task in progress
E	Int 11..33	Job and Task Identifier
P	INTEGER	Time Spent for each Job
Z	Int. 1..3	Available Machines

Fig. 13. The 3×3 job-shop scheduling problem.

Table 4
Initial marking for the 3×3 job-shop.

3×3 J-S model	Place nodes		
	P_1	P_2	P_3
Initial marking	1'(10,0)@0 + 1'(20,0)@0 + 1'(30,0)@0	1'(10,1,7)@0 + 1'(10,2,8)@0 + 1'(11,2,4)@0 + 1'(12,1,7)@0 + 1'(12,3,4)@0 + 1'(13,8,1)@0 + 1'(20,1,6)@0 + 1'(20,2,5)@0 + 1'(21,2,4)@0 + 1'(21,3,2)@0 + 1'(22,1,6)@0 + 1'(23,1,3)@0 + 1'(23,2,2)@0 + 1'(24,8,1)@0 + 1'(30,1,8)@0 + 1'(30,3,5)@0 + 1'(31,2,2)@0 + 1'(32,2,6)@0 + 1'(32,3,4)@0 + 1'(33,1,4)@0 + 1'(33,2,2)@0 + 1'(34,1,2)@0 + 1'(34,3,3)@0 + 1'(35,8,1)@0	1'(1)@0 + 1'(2)@0 + 1'(3)@0

- *Discarded nodes after the initial evaluation*: This parameter measures how many times the algorithm discards an evaluated S-old node because its time stamps are worse than the time information of the S-old node stored in the SS.
- *Undecidable nodes*: This parameter measures how many times the algorithm must perform a depth exploration until the objective state is reached because the time stamp combination is a mixture of greater and lower values.
- *Updated nodes after depth explorations*: This parameter takes into account how many times after the depth explorations the evaluated node certainly produced better values. In such a case its time stamps are updated along with the ones from its successor nodes (maintaining time consistency).

3.1. Simple analysis

The two-step approach implemented solved the job-shops problems presented in the previous section. The time taken to solve the problems varies depending mainly on two aspects: the size of the SS and the evaluations performed in the second step.

The obtained makespan corresponds to the optimal makespan reported for these problems are 15, 428 and 55 time units for the 3×3, 5×5 and 6×6 respectively.

Table 6 presents the performance measures obtained when the successor nodes were evaluated without any criterion in the first phase of the algorithm. This data has been used as the basis to evaluate whether the heuristics proposed give better or worse results based on the number of operations performed by the coded algorithm.

Table 5
Objective marking for the 3×3 job-shop.

3×3 J-S model	Place nodes		
	P_1	P_2	P_3
Objective marking	$1'(14,8) + 1'(24,8) + 1'(34,8)$	$1'(10,1,7) + 1'(10,2,8) + 1'(11,2,4) + 1'(12,1,7) + 1'(12,3,4) + 1'(13,8,1) + 1'(20,1,6) + 1'(20,2,5) + 1'(21,2,4) + 1'(21,3,2) + 1'(22,1,6) + 1'(23,1,3) + 1'(23,2,2) + 1'(24,8,1) + 1'(30,1,8) + 1'(30,3,5) + 1'(31,2,2) + 1'(32,2,6) + 1'(32,3,4) + 1'(33,1,4) + 1'(33,2,2) + 1'(34,1,2) + 1'(34,3,3) + 1'(35,8,1)$	$1'(1) + 1'(2) + 1'(3)$

Table 6
Performance evaluation with no heuristic implemented.

Job-shop type	J-S 3×3	J-S 5×5	J-S 6×6
Number of different states	693	7,776	117,650
S-old nodes found	2,032	24,625	487,408
Updated nodes at initial evaluation	59	1,893	26,555
Discarded nodes after initial evaluation	485	5,829	128,387
Nodes unable to decide	1,488	16,903	332,475
Updated nodes after depth explorations	11	10	35

Table 7
Results of the DFS algorithm with absolute time values heuristic.

Job-shop type	J-S 3×3	J-S 5×5	J-S 6×6
Number of different states	693	7,776	117,650
S-old nodes found	2,032	24,625	487,408
Updated nodes at initial evaluation	0	1,330	18,725
Discarded nodes after initial evaluation	2,032	10,506	213,966
Nodes unable to decide	0	12,789	254,717
Updated nodes after depth explorations	0	22	34

Table 8
Results of the DFS algorithm with a probabilistic heuristic.

Job-shop type	J-S 3×3	J-S 5×5	J-S 6×6
Number of different states	693	7,776	117,650
S-old nodes found	2032	24,625	487,408
Updated nodes at initial evaluation	62	1,272	16,734
Discarded nodes after initial evaluation	610	10,974	233,476
Nodes unable to decide	1360	12,379	237,198
Updated nodes after depth explorations	4	21	30

3.2. Heuristic based on absolute time values

In this case the J-S's were solved with the heuristic presented in Section 2.5.1, the obtained result corresponded to the same optimal solution but, the optimal was obtained faster than the initial procedure as it will be shown in Table 7. The heuristic guided the DFS algorithm for selecting the subsequent states in the different levels of the SS. The optimization results are presented in Table 7.

It can be seen from row 5 that the quantity of evaluations is drastically reduced compared to the evaluations made without any heuristic. The same result can be seen in row 6 in which the number of nodes was reduced by almost 80,000 implying that the quantity of depth explorations avoided led to the correspondent reduction in time getting the final results.

3.3. Heuristic based on probabilistic values

The heuristic based on probabilistic values, Section 2.5.2, was implemented in the first step of the algorithm, and the results are presented in Table 8.

Using this approach the job-shops were solved obtaining the optimal solutions and the number of key operations such as *nodes unable to decide* were reduced which makes the convergence to the optimal faster than the previous two implementations. The previous result suggests that it is possible to reduce the quantity of operations to get to the optimal scheduling depending on the way the SS is generated. It also can be seen that for a system with a reduced amount of states, such as the 3×3, the results are worse than the ones obtained in Section 3.2, but for systems with a considerable amount of states, which are the wanted ones to be solved, good performance is given. In the case of the 6×6 the two-step approach reduces the quantity of undecided outcomes (from 254,717 using the absolute time approach to 237,198), which improves the optimization phase and the overall performance of the algorithm.

4. Conclusions and future work

In this work, a state space approach, which deals with Timed CPN models applied to manufacture systems when the time component is a key factor to optimize the overall processing time, has been reviewed.

In the first part of the paper the modelling formalism together with a state space analysis approach is presented and discussed. They have been introduced the S-old nodes which are useful for developing approaches that avoid the storing and exploration of all the states in the timed state space. The use of the S-old nodes is properly described together with the time stamp's analysis of the repeated states to preserve time consistency in those states that must be explored to reach optimal solutions.

In the second part of the paper a two-step approach for the generation, verification and optimization of TCPN models is presented. The proposed phases separate the state space generation from the time optimization task. The presented approach was coded and evaluated with three job-shop problems, which appear to be good for testing the efficiency of the coded algorithm. The approach can also be improved by two heuristics, which allow in the first phase a better selection of states. In the second phase the optimization is carried out analyzing the time values of the repeated S-old nodes. The obtained results show clearly that the performance of the DFS algorithm in the first phase is crucial for a better performance of the second phase and for the minimization of computational burden. In this paper the time factor is used as the only factor to be optimized.

The presented approach was experimentally verified testing three job-shop problems. The results obtained corresponded to the optimal ones, it can not be theoretically ensured that the same behaviour will hold with different problems, reason why further research must be done in order to find the conditions that ensure optimality for this approach.

In the future other utility measures will be analyzed in order to apply the algorithm to other fields beside manufacturing and logistics. The two-step approach is a novel one for state space analysis and can be further improved at the two steps. Effective heuristics are presented in this paper, but challenging research is still an open area to improve the analysis of the old node list.

References

[1] ARENA Homepage. <http://www.arenasimulation.com/>.
[2] S. Christensen, T. Mailund, A generalized sweep-line method for safety properties, in: Proceedings of International Symposium of Formal Methods Europe, Springer, Berlin, Heidelberg, 2002, pp. 549–567.
[3] CPNTools Homepage. <http://wiki.daimi.au.dk/cpntools/_home.wiki>.
[4] S. Dauzére-Peres, J.B. Lasserre, An integrated approach in production planning and scheduling, in: Lecture Notes in Economic and Mathematical Systems, Springer-Verlag, Berlin, 1994.
[5] A. Gambin, M.A. Piera, D. Riera, A petri nets based object oriented tool for the scheduling of stochastic flexible manufacturing systems, in: Proceedings of the Seventh IEEE International Conference on Emerging Technologies and Factory Automation, vol. 2, 1999, pp. 1091–1098.
[6] C.R. Harrell, B.K. Ghosh, R.O. Bowden, Simulation Using Promodel, third ed., McGRAW-HILL, New York, 2000.
[7] G.J. Holzmann, An analysis of Bitstate Hashing, Formal Methods in System Design 13 (3) (1998) 301–314.
[8] C. Jard, T. Jeron, Bounded–memory algorithms for verification on-the fly, in: Proceedings of CAV 1991, Lecture Notes in Computer Science, vol. 575, Springer-Verlag, 1992, pp. 192–202.
[9] K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, vol. 1, Springer-Verlag, Berlin, 1997.
[10] K. Jensen, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, vol. 2, Springer-Verlag, Berlin, 1997.

[11] K. Jensen, T. Mailund, L.M. Kristensen, State space methods for timed coloured petri nets, in: Proceedings of Second International Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin, 2001, pp. 33–42.
[12] K. Jensen, L.M. Kristensen, Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer, Berlin, Heidelberg, 2009.
[13] Y. Merkuryev, G. Merkuryeva, M.A. Piera, T. Guasch, Simulation-Based Case Studies In Logistics: Education and Applied Research, Springer, 2009.
[14] K.E. Moore, S.M. Gupta, Petri net models of flexible and automated manufacturing systems: a survey, International Journal of Production Research 34 (11) (1996) 3001–3035.
[15] M. Mujica, M.A. Piera, Improvements for a coloured petri net simulator, in: Proceedings of the Sixth EUROSIM Congress on Modelling and Simulation, Ljubljana, Slovenia, 2007, p. 237.
[16] M. Narciso, M.A. Piera, J. Figueras, Optimización de Sistemas Logísticos Mediante Simulación: Una Metodología Basada en Redes de Petri Coloreadas, Revista Iberoamericana de Automática e Informática Industrial 2 (4) (2005) 54–65. Valencia, España.
[17] OptQuest. <http://www.arenasimulation.com/Products_OptQuest.aspx>.
[18] PetriSimM Homepage. <http://seth.asc.tuwien.ac.at/petrisimm/>.
[19] M.A. Piera, G. Music, Coloured petri nets: timed state space exploration examples and some simulation shortages, in: Proceedings of the MathMod 09, Viena, Austria, 2009.
[20] PROMODEL Homepage. <http://www.promodel.com/products/promodel/>.
[21] SimRunner. <http://www.promodel.com/products/simrunner/>.
[22] A. Valmari, The state explosion problem, Lecture Notes in Computer Science, vol. 1491, Springer-Verlag, London, 1998, pp. 429–528.
[23] Witness Homepage. <http://www.lanner.com/en/witness.cfm>.