

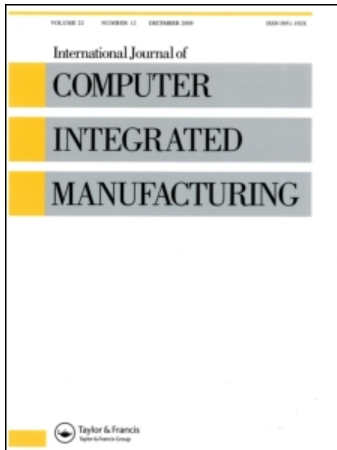
This article was downloaded by: [*Consorti de Biblioteques Universitaries de Catalunya*]

On: 20 January 2011

Access details: *Access Details: [subscription number 919083339]*

Publisher *Taylor & Francis*

Informa Ltd Registered in England and Wales Registered Number: 1072954 Registered office: Mortimer House, 37-41 Mortimer Street, London W1T 3JH, UK



## International Journal of Computer Integrated Manufacturing

Publication details, including instructions for authors and subscription information:

<http://www.informaworld.com/smpp/title~content=t713804665>

### **A compact timed state space approach for the analysis of manufacturing systems: key algorithmic improvements**

Miguel M. Mota<sup>a</sup>; Miquel A. Piera<sup>a</sup>

<sup>a</sup> Department of Telecommunications and Systems Engineering, Universitat Autònoma de Barcelona, Campus Universitari, Bellaterra, Barcelona, Spain

Online publication date: 15 January 2011

**To cite this Article** Mota, Miguel M. and Piera, Miquel A.(2011) 'A compact timed state space approach for the analysis of manufacturing systems: key algorithmic improvements', *International Journal of Computer Integrated Manufacturing*, 24: 2, 135 – 153

**To link to this Article:** DOI: 10.1080/0951192X.2010.543153

**URL:** <http://dx.doi.org/10.1080/0951192X.2010.543153>

PLEASE SCROLL DOWN FOR ARTICLE

Full terms and conditions of use: <http://www.informaworld.com/terms-and-conditions-of-access.pdf>

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

The publisher does not give any warranty express or implied or make any representation that the contents will be complete or accurate or up to date. The accuracy of any instructions, formulae and drug doses should be independently verified with primary sources. The publisher shall not be liable for any loss, actions, claims, proceedings, demand or costs or damages whatsoever or howsoever caused arising directly or indirectly in connection with or arising out of the use of this material.

## A compact timed state space approach for the analysis of manufacturing systems: key algorithmic improvements

Miguel M. Mota\* and Miquel A. Piera

Department of Telecommunications and Systems Engineering, Universitat Autònoma de Barcelona, Campus Universitari,  
Edif. Q, 08193, Bellaterra, Barcelona, Spain

(Received 15 November 2009; final version received 23 November 2010)

The state space (SS) analysis of a timed coloured Petri net (TCPN) has been used traditionally for validation and verification of system properties. Performance modelling using TCPN has also received the widespread attention of researchers in recent years as a promising alternative to improve productivity and competitiveness of present flexible manufacturing systems. In this article, a new computationally effective approach is introduced for designing efficient decision support tools based on the analysis of SSs, in which the computational time is an important requirement to deal with optimal scheduling, routing or planning policies. The SS analysis of a system specified in the TCPN formalism is faced with an algorithm in two stages, key implementation algorithmic aspects are considered to improve the time consuming tasks (transition evaluation, data management and information search). In order to provide good benchmarking results when applied to the optimisation of industrial scheduling problems, some examples are given and future work is addressed at the end of the article.

**Keywords:** timed coloured Petri nets; state space; simulation; decision support tools; optimisation; manufacturing; scheduling

### 1. Introduction

In the design of decision support tools for manufacturing systems, it is a common practice to analyse different system configurations by means of a simulation model that is the so called simulation-based optimisation approach. Unfortunately, this approach has the drawback of only covering a small set of all possible scenarios of a flexible system, which does not ensure reaching the optimal scenario configuration. In most of the cases, the reliability on the decision taken is only ensured with multiple experiments or simulation runs of the system's model.

Given its capabilities to properly model the behaviour of discrete event systems, the Petri net formalism has been extensively used during many years by the scientific community to develop manufacture and industrial models in order to determine several aspects of the modelled system. These aspects range from some design shortages of the real system through the analysis of the models properties (Liu and Wu 1993), to efficient policies to manage and control the resources of a floor-shop (Tchako *et al.* 1994) for improving some key performance indicators (Gradišar and Mušič 2007), amongst others. From some years ago, a focus has been put on employing Petri net models to cope with interesting managerial point-of-view problems such as

the scheduling problems of flexible manufacturing systems (FMS) (Lee and DiCesare 1994). Furthermore, some authors have devised approaches that put together the capabilities of the simulation approaches with heuristics and meta-heuristics parameterising a subset of decision variables within the model and then optimising the allocation of the variables through the use of the heuristics (Yu *et al.* 2003). The characteristics of flexible systems are suited to be modelled with the Petri net formalism without losing in the abstraction process important details that must be taken into account (Reyes *et al.* 2002); however, the use of this approach sometimes produces models with a similar complexity level as the original system, which hinders the understanding of the cause and effect relationships.

Recently, the use of high-level modelling formalisms such as coloured Petri nets (CPN) or timed coloured Petri nets (TCPN) have been introduced to model FMS problems in a higher abstraction level, allowing not only to model the key attributes that affect the system performance but also to facilitate the understanding of the cause and effect relationships of the original system. Furthermore, these formalisms can be used together with heuristics to develop optimisation approaches that are able to find optimal or close to the optimal configurations for these types of systems

\*Corresponding author. Email: miguelantonio.mujica@uab.cat

(Dashora *et al.* 2008, Mujica *et al.* 2010). In particular, the timed version of the formalism (TCPN) has a simple but efficient clock policy that permits to properly model the concurrent behaviour of systems. The use of the reachability graph or state space (SS) together with the model allows exploring the different configurations of the studied system. This approach is capable of ensuring optimal configurations when it is allowed to explore the complete SS of the timed coloured Petri net model. The main problem with this approach is the state explosion (Valmari 1998) which saturates computer memory, and consequently in most of the cases the complete SS can neither be generated nor evaluated. This is the reason why there is the necessity for developing alternative approaches which allow analysing greater SSs and allow the implementation of search techniques to explore the SS in a more efficient way, i.e. heuristics. The authors have developed an approach that uses TCPN models together with a compact version of the timed SS to deal with the optimisation of industrial problems where the multiplicity of states has been carefully analysed in order to propose reliable solutions.

### 1.1. Timed coloured Petri nets

CPN is a simple yet powerful modelling formalism, which allows the modelling of complex systems characterised by an asynchronous, parallel or concurrent behaviour (Jensen 1997a).

In order to investigate the key performance indicator values 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, i.e. TCPN. This extension is made by introducing a global clock representing the model time, time stamps which determine token availability (Jensen 1997a,b) and time consumption associated to the model transitions. Taking into account all of these elements the evolution of systems can be properly modelled through time stamp-updating when the transition firing takes place (Mujica *et al.* 2010).

The formal definition of the TCPN is as follows.

*Definition 1.* The non-hierarchical TCPN can be defined by a tuple:

$TCPN=(P, T, A, \Sigma, V, C, G, E, D, I)$  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)  $\Sigma$  is a finite set of non-empty colour sets.

- (5)  $V$  is a finite set of typed variables such that  $Type [v] \in \Sigma$  for all variables  $v \in V$ .
- (6)  $C: P \rightarrow \Sigma$  is a colour set function assigning a colour set to each place.
- (7)  $G: T \rightarrow EXPR$  is a guard function assigning a guard to each transition  $T$  such that

$Type [G(T)]=Boolean.$

- (8)  $E: A \rightarrow EXPR$  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)  $D: T \rightarrow EXPR$  is a transition expression which specifies the time delay associated to the transition. This time delay is generally denoted with the @ sign.
- (10)  $I$  is an initialisation function assigning an initial timed marking to each place  $p$  such that:

$Type [I(p)]=C(p)$

$EXPR$  denotes the expressions provided by the inscription language, and  $TYPE[e]$  denotes the type of an expression  $e \in EXPR$ , i.e. the type of values obtained when evaluating  $e$ . The set of variables in an expression  $e$  is denoted  $VAR[e]$  and the type of a variable  $v$  is denoted  $TYPE[v]$ .

*Definition 2.* The *untimed marking*  $M^U$  of a TCPN model is a function  $M^U: P \rightarrow EXPR$  that maps each place  $p$  into a multi set of values 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 3.* The *timed marking* of a TCPN is a function  $M^T: P \rightarrow EXPR$  such that  $M^T(p) \in C(p)$ . It maps each place  $p$  into a multi set of values  $M^T(p)$  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 (timestamps).

A token is *ready* if the correspondent timestamp 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.

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

*Definition 5.* The *input place nodes* of a transition are the ones whose directed arcs end in the transition.

*Definition 6.* The *output place nodes* of a transition are those in which the directed arcs of the transition end.

It is said that a transition is *enabled* if it is possible to find a binding of the variables that appear in the surrounding arc expressions of the transition such that the arc expression of each input arc evaluates to a multi-set of token colours that are present on the corresponding input place and the correspondent tokens are ready.

Each time a transition is *fired* it will remove from each input place the multi-set of tokens to which the corresponding input arc expression evaluates. Analogously, it will add to each output place the multi-set of tokens to which the corresponding output arc expression evaluates and it will attach timestamps to the created tokens. Following these rules, systems' evolution is modelled with token consumption and creation together with the time elements when a transition is *fired*.

To illustrate these concepts let us consider the simple manufacture model of Figure 1.

The model represents a manufacture system in which raw materials can be processed by two different manufacturing resources. This could be the case of a production system with two machines which are being used by two different processes (left-hand side of the figure). The system can be properly described by the TCPN model illustrated at the right-hand side of the figure. Place P1 represents the raw materials to be processed; process #1 is modelled with transition T1

and T3, and place nodes P2, P3 and P6. Similarly process #2 is modelled by transitions T2 and T4 and place nodes P4, P5 and P7. The machine-availability is modelled by the tokens present on places P3 and P5. Transition T1 and T2 model the activity of transporting and processing the raw materials to the correspondent machines and these activities consume 1 and 10 time units, respectively. Transitions T3 and T4 represent the moment of unloading the machines by the dedicated robots of Figure 1; these activities consume different time depending on the type of raw material. If the raw material follows process #1 the unloading takes 'y' time units but if process #2 is followed it will consume 'z' time units. In the right-hand side of Figure 1, the TCPN model depicts the initial state of the system (global clock = 0) represented by the underlined expressions which depict the number of tokens present in each place node (the brackets represent the timestamps associated to the tokens). In the initial state the transition T1 and T2 are enabled since the tokens in place P1 together with the ones from places P3 and P5 are ready and satisfy the restrictions associated to transitions T1 and T2.

The colours, multi-sets and the variables used by the model of Figure 1 are described in Tables 1–3, respectively.

The colour 'Type' is used to differentiate the type of raw material; the 'PTime' colour records the information of the time consumption by the unloading process when the machine has finished with the material. For example the colour combination in one token  $1'(2,2,1)$  means that the material is of type 2, and if it follows process #1 the unloading process will take 2 time units

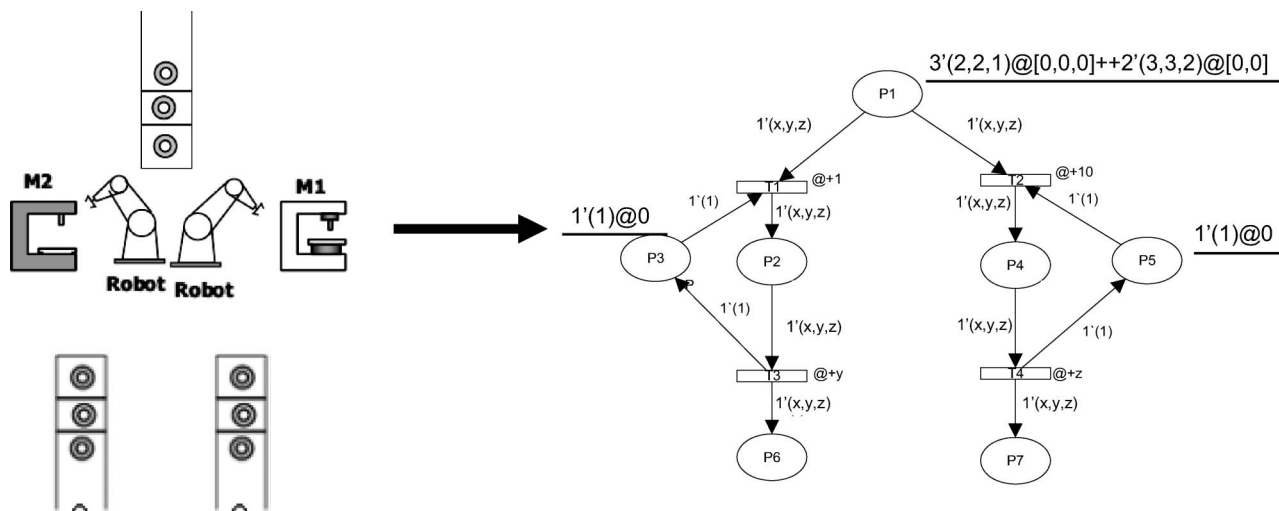


Figure 1. TCPN of a manufacture model.

while if it follows process #2 the unloading process will take 1 time unit.

Table 2 describes the multi-sets used in the different place nodes of the model and the type of colours used by the variables in the arc expressions are described in Table 3.

The dynamic behaviour of the TCPN model is illustrated in Figure 2. The right-hand side of the figure depicts the new state of the model once a firing has taken place.

The right-hand side of Figure 2 models the situation when one raw material has been used by process #1; in this case modelled by the token in place

Table 1. Colour definition of the manufacture model.

Colour sets	Definition	Description
Type	Type=integer	Describes the type of entity processed
PTime	PTime={1,2,3}	Describes the time elapsed to unload the machine
Constant	1	Describes the availability of the resource

Table 2. Multi-sets of the manufacture model.

Place nodes	Multisets	Description
P1,P2,P4, P6,P7	Product (Type × PTime × Ptime)	Describes the information of the processed entity
P3,P5	Constant	Describes the availability of the resource

Table 3. The variable set.

Variables
x: Type
y,z: PTIME

P2:  $1'(2,2,1)@1$ . This state represents that the raw material of type 2 has been used and the timestamp value of the token in place P2 means that this token will be ready again when the global clock reaches 1 time unit.

### 1.2. The compact timed state space

The SS or *reachability tree* (RT) of a timed CPN model is the set of all possible states or markings that can be reached from an initial one. Traditionally it has been used to verify CPN properties such as *reachability*, *boundedness*, *liveness* among others (Jensen and Kristensen 2009). In the SS of timed models, each node represents a timed marking of the TCPN model. Each node is connected with its successor nodes through directed arcs.

The main characteristics of the RT are:

- The root node represents the initial marking of the system.
- The successor nodes represent new markings once the firings have taken effect.
- For each node in the tree there must be generated as many successor nodes as enabling token combinations the marking has.
- The connecting arcs represent transition firings and they also have the information about the elements (tokens and transitions) that generated the firing.
- Different paths of the SS represent a series of different evolutions of the system.

During the generation of the SS, the TCPN model must interact with the reachability graph each time a new node is evaluated in order to determine which the successor states of the reached state are. This interaction is performed through the transition evaluation process of the evaluated node. Thus, the simulation

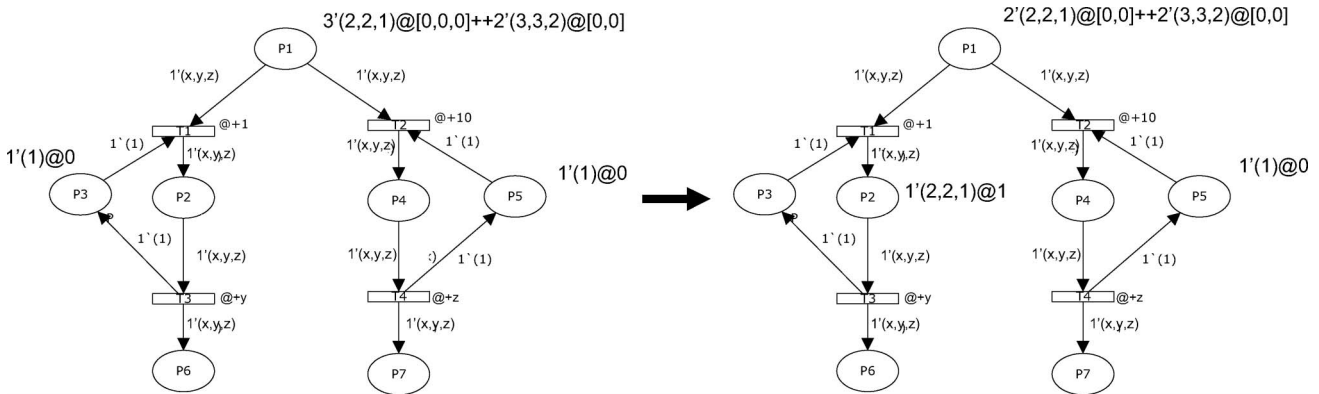


Figure 2. The dynamic behaviour of the TCPN model.

model interacts continuously with the reachability graph during its construction.

With the use of the SS it is possible to evaluate the different evolutions of a system; therefore the idea behind the use of the SS is to optimise the system's performance transforming the simulation-based optimisation approach into a search problem. Some authors have developed simulation-based optimisation approaches to optimise and verify system's performance (Lee and DiCesare 1994, Wells 2002, L alas *et al.* 2006) but unfortunately, under this approach, system configuration must be parameterised in such a way that any reachable state could be generated by a proper combination of model parameters, which usually requires a very complex logical model. This is usually not feasible in logistic systems, and the complexity is reduced only to a subset of the feasible configurations depending on the parameters they consider relevant for the study.

On the other hand, with the use of the reachability graph it is theoretically possible to explore all the states of the model. However in a real scenario, in order to avoid the computational burden, the model can be optimised by specifying a particular desired configuration and performing an exhaustive exploration with the use of a fitness factor to focus on the best solution that can be found considering computational time and memory restrictions. Although the idea sounds reasonable, in flexible systems with a considerable amount of decision variables, the models unfortunately suffer from the state explosion problem (Valmari 1998), which constraint the exploration to only a partial exploration of the possible configurations due to saturation of computer resources during the generation of the SS. Due to its nature the SS analysis has the potential to obtain the best solutions when it is properly explored. This is the reason for developing different search algorithms which alleviate as much as possible the saturation of computer resources.

One way of storing the SS information of timed models without storing the complete SS is with the use of the so-called *compact timed state space* (CTSS) (Mujica *et al.* 2010). The following definitions are necessary to describe the CTSS.

*Definition 7.* A state will be considered as an *old node* if it is exactly the same (together with its time values) as one that had been previously generated in any other level of the SS.

*Definition 8.* A *dead marking* is a state that does not have any enabled transition.

*Definition 9.* A *new state* is a node that is neither a dead marking nor an old node.

*Definition 10.* A *feasible path* is a sequence of nodes that go from the root node or initial state to the objective marking by firing enabled transitions. Each feasible path represents one system evolution from all the possible configurations that end in the particular configuration.

In order to avoid the complete storage of the timed SS, the CTSS uses the symmetric old node concept instead of the old node.

*Definition 11.* Symmetric old node (S-old node)

The symmetric 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. Let us write this concept in a more formal way. Being  $\mathfrak{M}^T$  the set of timed markings of a SS. Let  $M_i^T$  and  $M_k^T$  be timed markings with their correspondent untimed markings  $m_i$  and  $m_k$ .

A marking  $M_i^T$  is an *S-old node* to another  $M_k^T$  marking when the following condition holds:

$$M_i^T, M_k^T \in \mathfrak{M}^T \wedge m_i = m_k$$

The use of the S-old node concept, contributes to mitigate the computer requirements (time and memory) because it is possible to avoid the full evaluation of those timed states whose underlying untimed markings are the same (Mujica *et al.* 2010). This is the case of stochastic timed systems where the untimed markings corresponds to a particular system's configuration but is reached at different times.

## 2. Computational efficiency: evaluation methods

In this section, the different elements are explained that one must efficiently implement in order to develop a computer tool that allows the use of the CTSS together with the TCPN formalism for facing optimisation problems.

The four main tasks or procedures that must be focused in order to develop an algorithm that generates, evaluates and optimises TCPN models efficiently are: transition evaluation, management of marking information, detection of repeated states and analysis of the S-old nodes. Since the last task is explained by Mujica *et al.* (2010) the first three elements of the algorithm are presented in the following sections in order to describe how the implementation has to be done for an approach based on the analysis of the CTSS.

### 2.1. Transition evaluation as a constraint satisfaction problem

The transition evaluation procedure is the one that finds the tokens that enables a transition. This

procedure takes into account all the restrictions that participate in this procedure: cardinalities of the arc expressions, variables in the arc expressions, constant values in the arc expressions, time restrictions, input places in the transition and the different guards in the transition. It must be noted that this procedure takes place in each transition of the model so the more tokens in the model the more time it will take the procedure to determine the enabled transitions in the model. It is fair to mention that the search procedures performed by the procedure in order to find the enabling tokens has nothing to do with the search procedures that take place in the SS. All the performed searches or procedures to find the enabling tokens are performed locally that is they take place in each transition of the model.

Since these procedures are performed for each transition, various authors have mentioned this procedure as one key factor that determines the overall performance of the simulation algorithms based on the CPN modelling formalism (Gaeta 1996, Mortensen 2001).

CPN has rules that impose restrictions to the transition, firing task (Jensen 1997a,b); these restrictions must be satisfied in order to generate the successor states. Different algorithms have been developed for this objective (Gaeta 1996, Mortensen 2001, Jensen and Kristensen 2009); most of these algorithms evaluate the expressions stated in the *guards* in a passive way, that is, the algorithm checks whether the chosen tokens satisfy the guard expressions after they have satisfied the restrictions imposed by the arcs. This way of evaluating the tokens is not efficient when the number of feasible token combinations is poor with respect to the total amount of tokens in the input places. By considering the arc and guard expressions as constraints, it is possible to mitigate the evaluation computational burden, using some principles of constraint programming to solve the restriction satisfaction problem.

The algorithm faces the problem of satisfying the restrictions associated to transitions as a constraint satisfaction problem (CSP) (Tsang 1993) is presented in this section and it consists of three main steps.

Step 1. The algorithm pre-processes the tokens in the place nodes of the CPN model to avoid computation of worthless token combinations that do not satisfy the arc restrictions. It constructs a sub-set of the original tokens in the input places satisfying the constant values present in the arc inscriptions. In this procedure, the algorithm does not take into account the variables present in the arc inscriptions, instead it uses them as wild cards allowing to select only the tokens that match the constant values of the arc inscriptions.

Step 2. When the pre-processing task has taken place, it uses the restrictions in the guards in an active way to solve the problem of finding the subset of tokens that satisfy the guard restrictions and enable the transitions of the CPN model.

Step 3. When the restrictions have been satisfied (arc inscriptions and guard restrictions) and the correspondent tokens found, the last step is to fire the transition with all the possible combinations of the tokens in the solution subsets.

Another important aspect of the transition evaluation algorithm is how to obtain the token sets that satisfy the guard restrictions. It has already been mentioned that the transition enabling procedure can be stated as a CSP. It is important to remark that the CSP is solved after the pre-processing task has taken place using only the sub-sets obtained from that procedure. In the example of Figure 3, the arc inscription  $1'(X,Y,4)$  will be used to exclude all the tokens that do not satisfy the constant value of the third colour (step1).

The elements in the CPN model have correspondent elements in the CSP as follows:

Z: the variable set that corresponds to the arc inscription variables of the input arcs of the transition.

D: the functions that map the colour tokens in the input places to variable values, in this particular case this is done with the identity function.

C: the restriction set that must be satisfied by the variables Z, this restriction set is given by the guard expressions in the transition.

Figure 3 is used to illustrate how the CSP is defined from a transition evaluation problem.

The arc inscriptions  $1'(X,Y,4)$  and  $1'(R,W)$  use the variables X,Y,R and W which will be used to define the variable set Z in the CSP:

$$Z = \{X, Y, R, W\}$$

The identity function:

$$\begin{aligned} F: Z &\rightarrow Z \\ p &\mapsto p \end{aligned} \quad (1)$$

is used for mapping the variable values from the token colours of the input places obtaining the variables domain of the correspondent variables, in which  $p$  corresponds to the token colours in the input place.

$$\begin{aligned} X &: \{3, 4, 5, 6\} \\ Y &: \{2, 3, 4, 6, 8\} \\ R &: \{2, 4, 5, 6, 7\} \\ W &: \{2, 3, 4, 6, 8\}. \end{aligned}$$

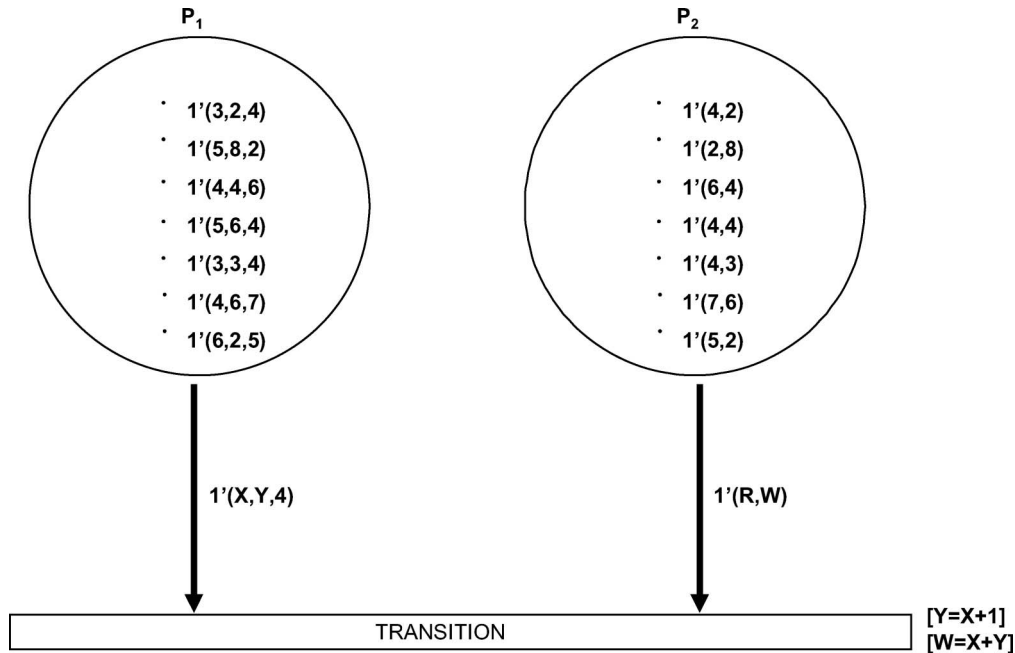


Figure 3. The transition evaluation task.

The restrictions that must be satisfied are obtained directly from the guard expressions:

$$C : \{Y = X + 1, W = X + Y\}.$$

When the transition evaluation problem is stated as a CSP it is possible to apply CSP solution algorithms to obtain the solutions to the CSP which correspond to a solution or set of solutions of the transition evaluation problem.

The solution of the CSP problem has correspondence with the tokens that enable the transition.

Algorithm 1 presents in pseudo code the implementation of the pre-processing of tokens and the solution to the CSP problem (steps 1 and 2).

Algorithm 1: Generation of valid sets according to transition arc restrictions.

```

FOR all input places DO
  MAKE_SET(MARKING, Arcs_AllVars)
NEXT
N:=1
Fp:= final place
FOR PLACE. List DO
  Token:=PLACE.list(N)
DO
  Place:=Place+1
  List_VarZ:= Find_VarZ ( transition,place)
  FOR all_Variable_set DO

```

```

IF Var.status=not binded THEN bind_
  value(Token)
  PROPAGATE_VALUE
  (Var.Restriction)
  MAKE_SET(MARKING, Arc, Var)
  If Size(Set)=0 then EXIT DO
NEXT
REPEAT UNTIL Place= Fp
N:=N+1
NEXT

```

The first loop makes the pre-processing of the tokens to satisfy the arc restrictions. It uses the function **MAKE\_SET** to construct subsets through filtering the initial tokens that satisfy the constant values of the arc expressions; it uses the model marking and the arc variables as arguments. The procedure carried out by **MAKE\_SET** is illustrated by Figure 4.

The **MAKE\_SET** function is used at the beginning of the algorithm to construct the initial sub set which satisfies the constant values of the arc inscriptions (i.e. tokens with value 4 in the third colour:  $1'(x,y,4)$ ).

The nested loops are used to solve the CSP through the binding of variable values from the token colours and the value propagation using the restriction set.

The inner loop uses the **PROPAGATE\_VALUE** procedure to bind the values of the arc expressions to the colours of the correspondent token. The binded variable and the restriction expression are used as arguments to propagate the variable value to the other



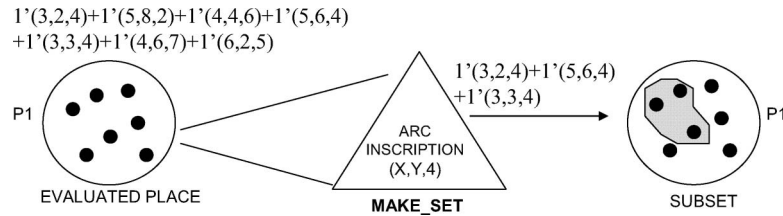


Figure 4. Generation of the initial set.

variables that are part of the restriction. When the value bindings take place, the initial subsets are reduced due to the restriction imposed by the value propagation in the correspondent place. If no subset can be obtained through value propagation  $\text{Size}(\text{Set})=0$  then a backtrack is carried out to select the next token from the previous place node.

The external loop is used to carry out this procedure in all the input place nodes until all the subset tokens have been evaluated. At the end of the algorithm, the sets that satisfy both the arcs expressions and the restrictions are found.

Figure 5 illustrates the procedure carried out by Algorithm 1.

The problem starts when no subsets have been developed (Figure 5a). The first loop uses the **MAKE\_SET** and it constructs the sets with those tokens that match the constant values of the arc expressions. As a result of the initial loop, the subsets that satisfy the arc expressions are obtained (Figure 5b). The nested loops solve the CSP using the initial subset as a starting point, and it progressively generates smaller subsets through value propagation and backtracking to obtain the final subsets that satisfy all the restrictions (Figure 5c), as it was explained before.

Once the tokens that enable the transition have been found, the final step is to fire the transition using all the possible token combinations to produce the successor nodes. With the use of this approach it is possible to reduce by 90% the number of operations needed to solve the transition evaluation problem as it will be shown later in the article.

## 2.2. State space data structure

As mentioned before, when dealing with SS analysis, an important computational problem is the state explosion problem (Valmari 1998). Therefore, it is necessary to develop approaches and data structures that alleviate as much as possible the amount of space used when storing all the markings of the RT.

Due to the locality principle (Mortensen 2001) every time a transition is fired, the marking information only changes in the input and output places to the fired transition. Due to this characteristic there are only small differences in the marking information before and after the firing, so the unchanged information (i.e. places not altered by the fired transition) is stored several times during a simulation producing a memory burden. This information redundancy can be taken into account to develop data structures that reduce the memory burden so that it is possible to store higher amounts of states.

In the following sub sections, a way of storing the markings by decomposing the marking data into two layers of information is presented so that it is not necessary to store all the marking information each time a new one is generated during the CPN simulation.

## 2.3. Place node information layer

The *colour layer* ( $C\_Layer$ ) of information is used for grouping the information related to the markings of each place node. The grouping uses two data blocks in which one data block groups the colour combination that appears in a place and the other stores the number of appearances (cardinalities) of the tokens with the same colour combination in the first block. In this layer the information of appeared cardinalities of a particular colour are related with the colour block. Therefore, different appearances of markings for a particular place are stored with the pairs  $\langle \text{cardinalities}, \text{colour} \rangle$ .

Figure 6 illustrates the structural relationship when the marking evolves in a typical SS.

When modelling industrial systems, it is typical to model resources such as raw materials, available machines, workers, etc. which could be natural to represent with coloured tokens in which the colour would be used to differentiate the kinds of resources. In those cases this approach is very useful to save memory during the generation and storing of the SS.

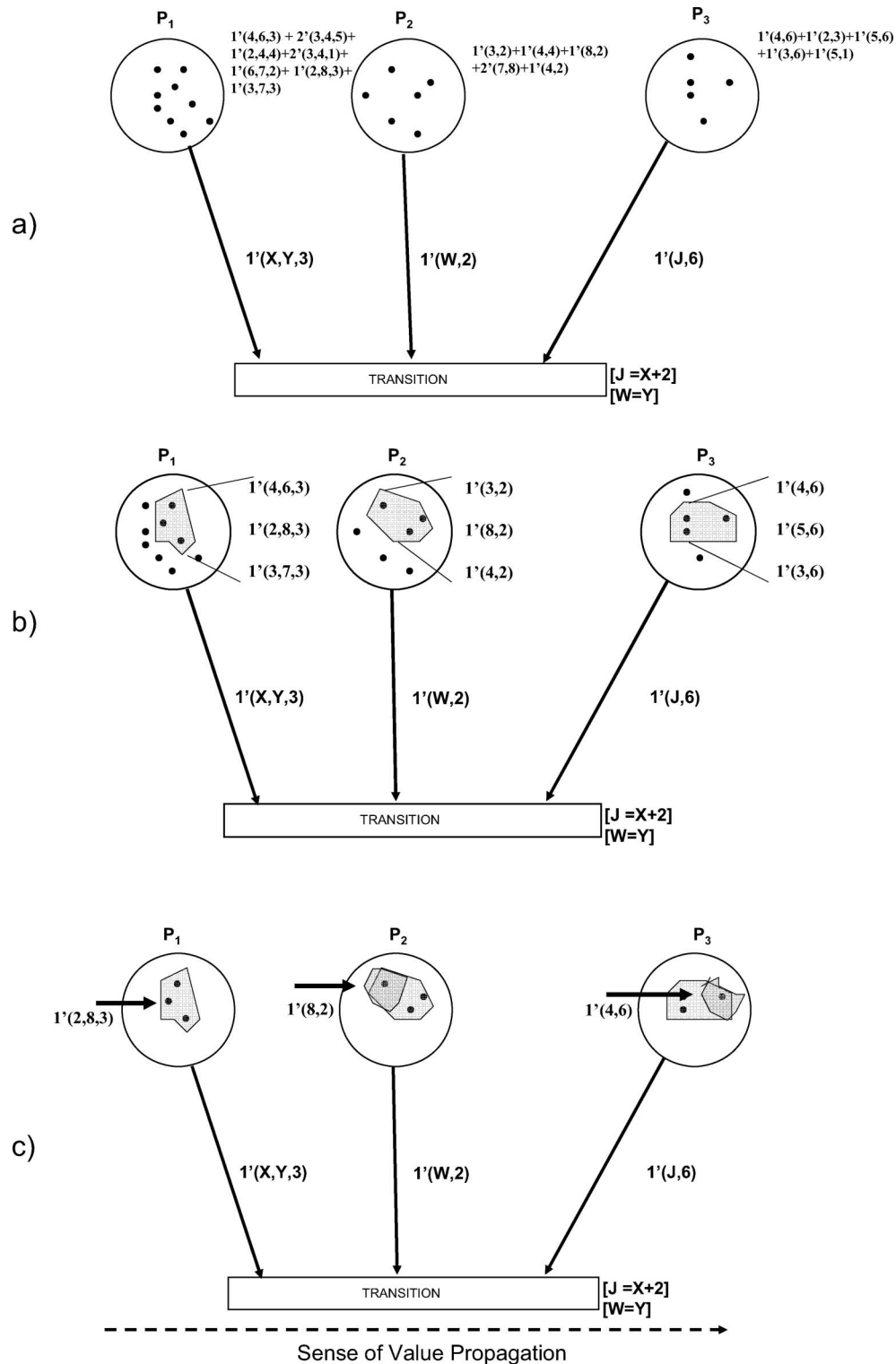


Figure 5. Generation of subsets through value propagation.

#### 2.4. Marking information layer

The second level of information, the *relationship layer* (R\_Layer), stores the relationship between the position of the place nodes in the C\_Layer and the cardinalities

used in the marking. The R\_Layer uses a data structure to specify the relationship between cardinalities and colours used in a place node and another data structure is used to specify the marking through the

relationship between places. Figure 7 illustrates the relationship between the two layers and the kind of information stored at the R\_Layer.

Using this relationship, any marking that appears in the model can be represented with the tuple  $\langle \text{cardinalities used, relation chain} \rangle$ . In addition to the relation chain, the R\_Layer stores also information about the SS structure like parent–children relationship, old nodes, children of a node and time attributes.

**2.5. Management of the S-old nodes**

The detection of new or repeated states is another important factor when the search-based optimisation approach is implemented. Since the amount of memory resources is finite, it is essential not to duplicate information. Hence, it is very important to identify whether a state has already appeared or not in the SS.

Because there is a strong relationship between the C\_Layer and the R\_Layer, it is necessary to establish an internal search algorithm to detect new or repeated

states that takes into account these two layers giving an efficient performance when dealing with great amounts of information.

The internal search algorithms implemented in the different blocks of information are based on binary search. It is known that, although it is an old algorithm, binary search presents certain advantages (Gonnet 1984) that are appropriate for the objectives pursued within this work:

- Computational complexity  $O(\log_2(N))$
- It maintains a logarithmic order if it is used only to compare values
- It is efficient to search data ranges or data values
- It is a stable algorithm, i.e. the search time range keeps close to the average search time, and the variance of search times is  $O(1)$ .

**2.6. The new-state detection**

Using the two layers of information for the search of repeated states, it is possible to save some CPU time

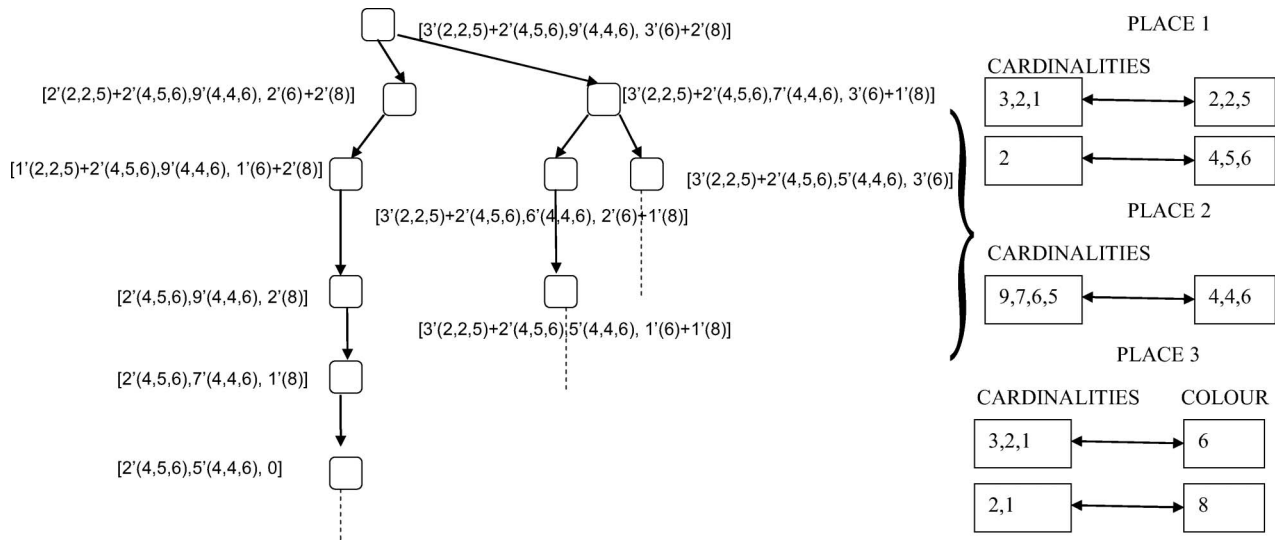


Figure 6. Storing place marking information using two blocks of data.

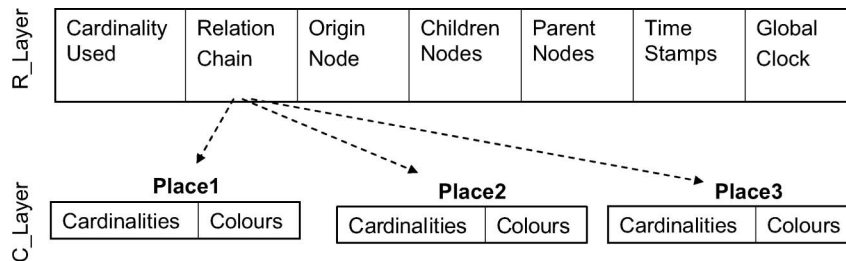


Figure 7. Relationship between the two layers.

when determining whether a state is a new one or not. Each time a transition is fired, the marking produced in a place is searched (using the binary search) at the C\_Layer to check if the place marking has already appeared. If a place marking at the C\_Layer is a new marking then the marking of the whole TCPN model is a new one.

When one new place marking has been found at the C\_Layer it is not necessary to search any further at the R\_Layer and the relationship between places is directly stored at the R\_Layer.

Algorithm 2 presents in pseudo code the search algorithm used for detecting new states each time a transition is fired.

Algorithm 2: Detection of new states

```

MARKING.STATUS := NEW
X:=0
REPEAT
  X=X+1
  Place=Output_Place (X)
  Colour=Fire_transition(MARKING,Place)
  In_Place_List= FIND_COL (Colour, Place)
  IF In_Place_List= TRUE THEN
    MARKING.STATUS=UNKNOWN
  END IF
  UNTIL Place=Final_OUTPUT PLACE
  IF MARKING.STATUS=NEW THEN
    ADD_MARKING ( MARKING)
  ELSE
    In_Marking=FIND_MARK(MARKING)
    IF IN_Marking=TRUE THEN
      Add_List_SOLDNODES (MARKING)
    ELSE
      Add_RT_LIST(MARKING)
    END IF
  END IF
END IF

```

The algorithm consists of one loop which evaluates the changes in the place nodes at the C\_Layer once the firing takes place. It checks whether the place marking already exists in the stored list.

The **FIND\_COL** procedure makes the binary search at the place list in the C\_Layer indicated by the argument *Place*. If the procedure determines that the colour combination is a new one then it is stored in the list of the corresponding place at the C\_Layer, and the Boolean variable **In\_Place\_List** takes value FALSE. On the contrary if the colours combination is a repeated one, it is not stored in the corresponding place and the Boolean variable **In\_Place\_List** takes the value FALSE and the MARKING.STATUS takes the value UNKNOWN.

The **FIND\_MARK** procedure uses the binary search at the R\_Layer to search for the marking

when the marking status is UNKNOWN. In such a case the search is done to check whether the marking is a new marking or a repeated one. If the marking status is NEW then the marking is directly added to the marking list at the R\_Layer with the procedure Add\_RT\_LIST but if the search determines a repeated S-old node (repeated marking in the CTSS) then its time elements are added to the S-old node list using the procedure Add\_RT\_LIST.

### 3. Computational performance

The algorithms presented in the previous sections were implemented for a SS analysis and optimisation approach which analyses the CTSS in two phases. The first phase generates the CTSS and the second phase analyses the repeated S-old nodes to optimise the feasible path that goes from the initial state to a particular one.

#### 3.1. Exploring the CTSS for scheduling objectives

In Narciso *et al.* (2009) the approach is addressed that uses the CPN SS as a search space for finding feasible paths starting from an initial state and ending at a particular one through exhaustive search. A variation of this approach is presented in Mujica *et al.* (2010) using the CTSS for optimising the makespan of manufacture models in two stages. In the first step of the approach, it generates the CTSS and it stores the S-old nodes that appear during the generation phase in a separated list for later analysis. In the first step, a Depth First Search algorithm is used for the generation of the different nodes in the tree and it generates also a feasible path that goes from the initial state to the objective one. Figure 8 illustrates this optimisation approach. Starting from an initial state, the different nodes of the tree are generated. The shaded nodes in the figure represent the S-old nodes found during the generation phase. The dot-line represents the feasible path found during the first phase of the algorithm.

In the second step, the S-old nodes are analysed by focusing on the timestamp values of the repeated S-old nodes to progressively optimise the feasible path. In Mujica *et al.* (2010), the procedure to analyse the time values to determine which branch of the tree produces better time values when an S-old node is found is detailed. Figure 9 shows the flow diagram of the optimisation algorithm.

The use of the S-old nodes allows minimising of the state explosion problem, since it is possible to avoid the storing of the sub trees in the SS that hang from the repeated S-old nodes. Instead, it stores only one sub-tree with the time values of the S-old node that appeared first in the generation step. The advantage of analysing the

SS in two steps is that it allows to implement heuristics in both phases that focus on the best way of performing the correspondent phase, i.e. if the first phase is taking place, the heuristic must select the best child nodes that potentially improve the second step, and the second step can be improved by implementing intelligent analysis methods to select the best repeated S-old nodes to be analysed. Other approaches based on TCPN have been developed (Störrle 1998) but they have been implemented to simulate and verify model properties that determine the behaviour of the model. The use of the approach presented here allows property verification in the first step, and the second step is used only for optimisation purposes.

3.2. State space data information storage

An academic example of a manufacture process where some resources are shared has been coded in order to evaluate the amount of memory spent in SS storing and the benefits obtained with the implementation of the CTSS. Figure 10 presents the TCPN model developed for this purpose.

Place node P1 models the raw materials or entities to be processed. Transition T1 represents the starting of a process which uses a processing machine modelled by the available token in place P3. Once the process has started, a token is generated to place P2 and one token is extracted from place P1 and another one from place P3. The firing of transition T3

models the end of the manufacture process and it generates a token to place P3 (modelling the machine availability) and another token to place P6 which models the processed material. A similar process is modelled with transitions T2 and T4 and places P4, P5 and P7, but they only differ in the time consumption to process the materials.

Several SSs were generated for the same manufacture model, varying only the amount of initial products to be processed. The SS information was stored in a spreadsheet in order to evaluate the amount of memory reduction achieved with the proposed implementation. Table 4 summarises the information of the evaluated models, and the correspondent SS information.

With the use of CTSS the amount of generated states is considerably less than the one with a typical TSS. The use of the compact representation of the timed SS presents a great advantage with systems which present big SS, in which the S-old nodes allow avoidance of the storing of sub trees that increase the memory usage. Figure 11 shows a comparison of the memory usage between a typical SS and a CTSS when the SS grows for the same model, the figure has been plotted in a semi-log scale in order to appreciate the linear growth (exponential in a normal scale) of both approaches.

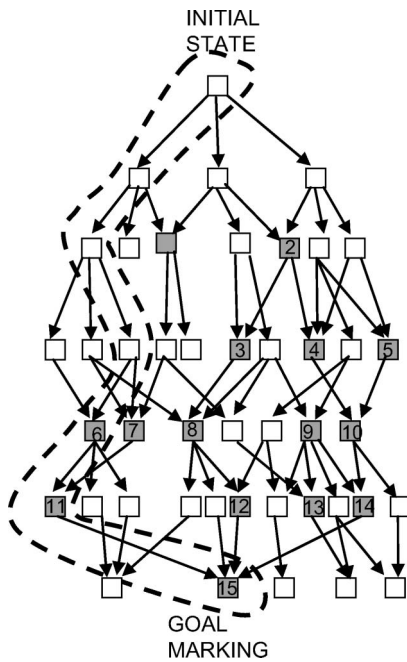


Figure 8. A feasible path when exploring the SS.

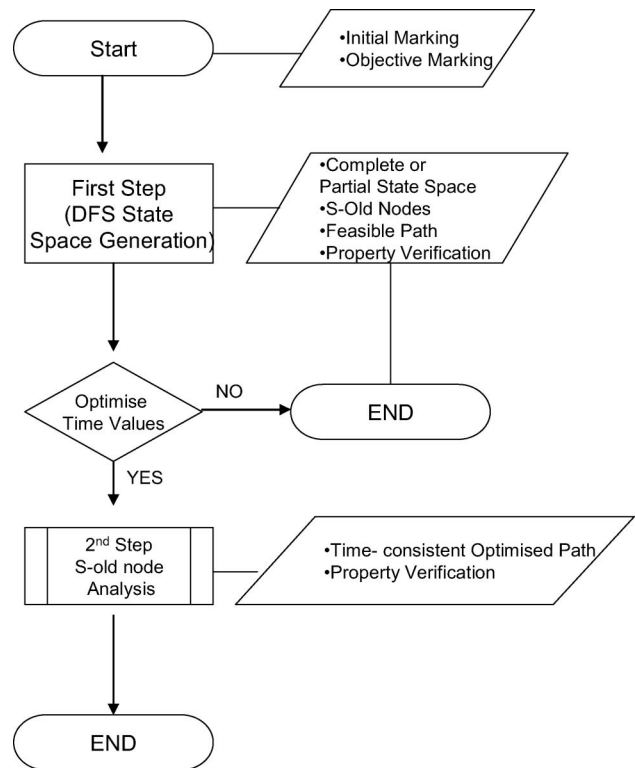


Figure 9. Flow chart of the generation/optimisation algorithm.

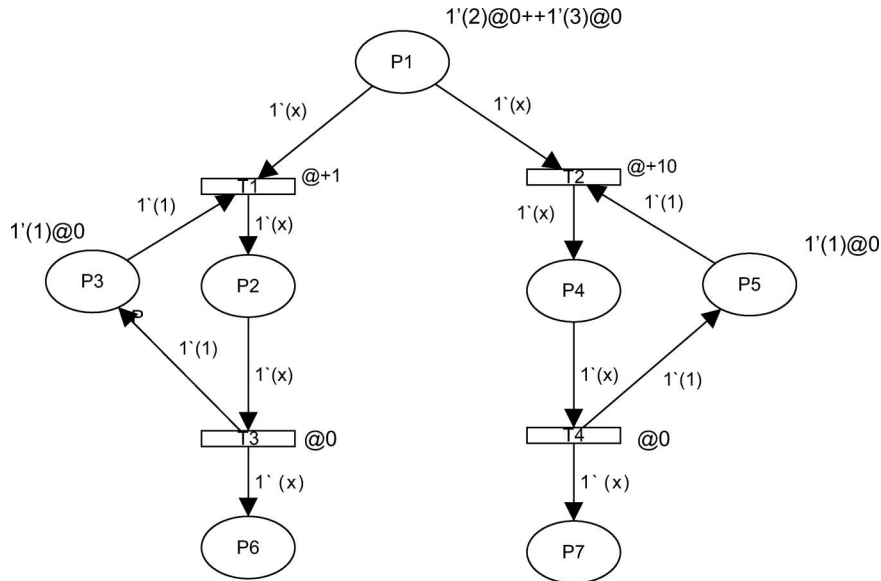


Figure 10. Manufacture model.

Table 4. Storage comparison between the CTSS and a typical TSS.

No. products	Initial tokens (P1)	Estimated number of nodes in the TSS	Kbytes used (kb)	Actual number of nodes in the CTSS	Kbytes used (kb)	Reduction obtained (%)
8	$4'(2)+4'(3)$	1820	253	1205	244	3.5
10	$5'(2)+5'(3)$	6660	931	2571	5194	44.25
14	$7'(2)+7'(3)$	130700	16,550	8408	17084	89.67
20	$10'(2)+10'(3)$	1,40,0000	177300	30866	6444-kb	96.36

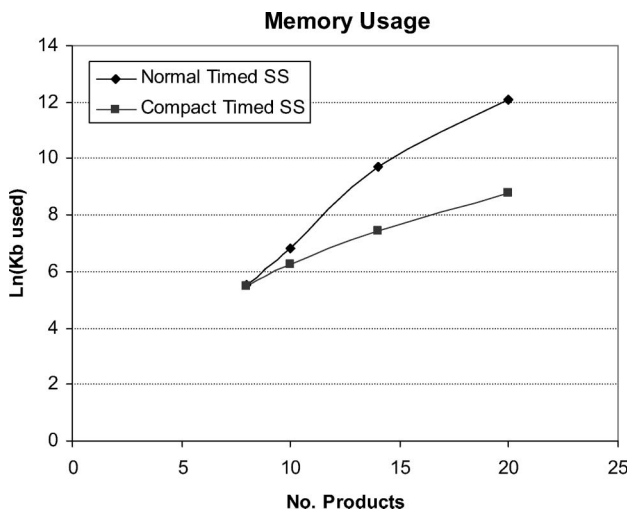


Figure 11. Memory usage for the CPN manufacture model.

It can be appreciated that due to the combinatorial explosion, the size of the SS grows exponentially with the increase of entities to be processed which is also

reflected in the memory needed to store the complete SS. With the use of the CTSS the slope in the graph is smaller than one of the typical timed SS thus the exponential growth using the CTSS is considerably delayed and it allows evaluation of bigger SSs than the ones that could be evaluated using a typical SS.

### 3.3. Computational efficiency of transition evaluation

The natural way of evaluating the tokens that enable the transition is to develop all the possible combinations of the input place nodes and then test if a particular one satisfies or not the restrictions imposed by the guards. Based on that assumption, the quantity of operations performed for testing the token combinations can be calculated using the following formula

$$\prod_{i=1}^r P_{ij} \tag{2}$$

where  $P_{ij}$  is the number of tokens in the input place  $i$  for the transition  $j$ , with  $r$  input place nodes. The base

of formula (2) is straightforward because all the possible token combinations must be tested one-by-one to check their validity. The formula just calculates the permutations of the tokens present in the input place nodes.

For the case of the CSP algorithm for the transition evaluation task, the number of operations performed to obtain the tokens combination that enable the transition can be calculated with the following formula

$$\sum_{i=1}^r T_{ij} + \prod_{i=1}^r S_{ij} \tag{3}$$

where  $T_{ij}$  is the total number of tokens present in the input place  $i$  for the transition  $j$ . The first term of formula (3) evaluates all the tokens one-by-one in the input place nodes to determine those that do not violate the initial arc restrictions (pre-processing task of the input place nodes). After the pre-processing task has taken place, the obtained subset of tokens  $S$  are those that satisfy the constant values of the arc inscriptions and the second term of formula (3) calculates the permutations of those subsets to find the final token combinations that enable the transition. From formula (3) it is easy to evaluate that the pre-processing task reduces considerably the original number of permutations necessary to determine the validity of the tokens present in the input place nodes.

In order to exemplify the calculations for both approaches, let us assume that there is a transition ( $j = 1$ ), that has three input places ( $r = 3$ ) with 50 tokens in each one (Figure 12).

In the case that the calculations were done using the natural approach (testing all the possible token combinations) the number of operations that would result is:

$$\prod_{i=1}^r P_{ij} = 50 \times 50 \times 50 = 125,000 \text{ operations.}$$

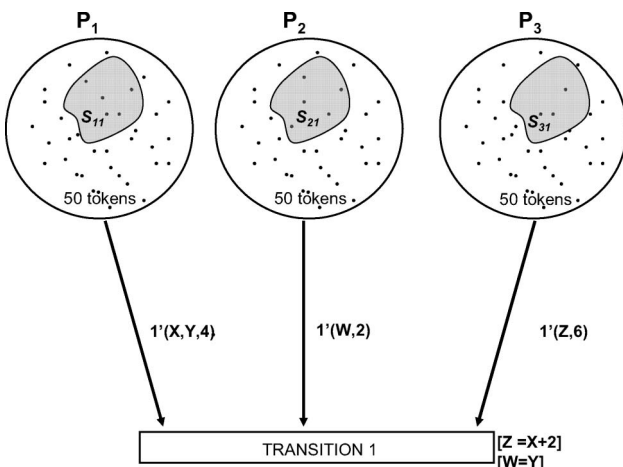


Figure 12. Transition evaluation example.

With the use of the developed algorithm and assuming that the pre-processing task produces sub-sets of 20 tokens each:

$\sum_{i=1}^r T_{ij} = 50 + 50 + 50 = 150$  operations the ones needed to obtain the initial sub-sets.

$\prod_{i=1}^r S_{ij} = 20 \times 20 \times 20 = 8000$ , the operations made by the CS algorithm in the worst case.

Based on those values, the reduction obtained in this particular example can be computed as:

$$\text{Reduction} = (125,000 - 8150) / 125,000 \times 100 = 93\%.$$

### 3.4. Generation and detection of states and overall efficiency

Since the state generation task depends also on the availability to detect the repeated or new nodes, the efficiency performance of these two activities has been verified through the evaluation of a very well-known benchmark problems, the  $3 \times 3$ ,  $5 \times 5$  and  $6 \times 6$  job-shops.

#### 3.4.1. Job-shop

The job-shops in its different modalities (Dauzère-Peres and Laserre 1994) are a group of benchmark problems which consist in a certain number of jobs that must follow a specified sequence of tasks through different machines. Due to its nature the scientific community for several years has paid a lot of attention to solving these types of problems, which are considered the toughest to be solved (Dauzère-Peres and Laserre 1994, Lee and DiCesare 1994, Yogeswaran *et al.* 2009, Chan *et al.* 2010, Renna 2010). The PN modelling formalism possesses the

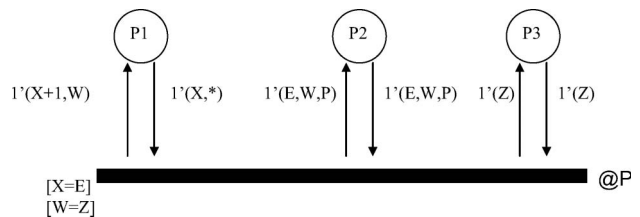


Figure 13. The timed CPN model of the  $3 \times 3$  job-shop.

Table 5. Colour definition of the job-shop  $3 \times 3$ .

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	Spent time by each job
Z	Int1.3	Available machines

necessary characteristics to model the complexity of these problems; therefore it has been widely used as a modelling formalism to cope with the allocation problem of these types of systems (Van der Aalst 1995, Adballah *et al.* 1998, Kumar *et al.* 2004). In the 3 × 3 job-shop three jobs must go through processes in three different machines. The goal of the study is to obtain the sequence that minimises the makespan of the whole procedure using the optimisation approach introduced in section 3.1.

The job-shops were modelled using the timed CPN formalism, the SS was generated implementing the algorithms presented in the previous sections and the different paths were automatically evaluated and analysed to find the one that optimised the makespan.

An example of a model developed for a job-shop is presented in Figure 13 which consists of one transition with three places.

The colours used in this model are presented in Table 5 with the correspondent description of the colour meaning.

Table 6 presents the multi-sets used in each place in the CPN model; there is also a description of the kind of information that is modelled within those places.

The P1 place stores the tokens which represent the information of job and task numbers through a composite colour X where the first digit represents

the job and the second digit the task number; as an example let us say that X has the value 13, it represents that the job 1 is performing task 3. Colour W is used to specify the machine needed to complete the correspondent task, so if W has the value 1 it represents that the machine needed for the next task is machine number 1.

The P2 place stores tokens which have colours used by the model to generate the output tokens that model the evolution of job and tasks of the job-shop. This evolution is modelled with the information stored in the colours of the tokens stored in place P2, together with the restrictions imposed by the guards. The first colour in place P2 (E) states the job and task needed (modelled in place P1) to generate the next token when the transition takes place. The output token will be generated using also the information of colour W of tokens in place P2. Once the fires take place the task identifier in the output token in place P1 will be incremented by 1 to model the new task and the machine needed for the next task will be modelled

Table 6. Colour sets definition.

Place	Multisets	Description
P1	Product X × W	The tokens in this place hold the information of which task is being processed at the moment and which machine is needed afterwards
P2	Product E × W × P	The tokens of this place hold the relationship between tasks and the time needed to complete each task
P3	Z	The tokens in this place represent the availability of machines for the different tasks.

Table 7. Initial marking of the 3 × 3 job-shop.

Job shop 3 × 3	Place node		
	P1	P2	P3
Initial	1'(10,0)@0	1'(10,1,7) @0	1'(1) @0
Marking	+1'(20,0)@0	+1'(10,2,8) @0	+1'(2) @0
	+1'(30,0)@0	+1'(11,2,4) @0	+1'(3) @0
		+1'(12,1,7) @0	
		+1'(12,3,4) @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'(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		

Table 8. Search efficiency between sequential and binary search.

Job shop type	Type of implemented search	Different nodes	S-old nodes found	Performed searches	Time consumed by the algorithm
3 × 3	Sequential search	693	680	10,204	5 min
3 × 3	Binary search	693	680	10,204	12 s
5 × 5	Sequential search	7,776	7,750	121,825	35 min
5 × 5	Binary search	7,776	7,750	121,825	7.5 min
6 × 6	Sequential search	117,650	117,612	2,302,000	5.5 h
6 × 6	Binary search	117,650	117,612	2,302,000	1.5 h



using colour W, e.g.  $1'(X+1,W)$ . The P colour of place P2 will be used by the transition to calculate the timestamp of the output token once the firing takes place (@P).

The tokens in place P3 model the availability of the different machines through the use of colour Z.

The initial marking is presented in Table 7. It models the situation when all jobs are waiting for the first task to start. For example, the token  $1'(10,0)@0$  in P1 represents job 1 in task 0 using no machine at time 0; the machine-availability at 0 time is modelled with the correspondent inscriptions '@0' in place P3.

The two other job-shops can be developed in a straightforward way using the example presented here.

3.4.2. Search efficiency test

In order to evaluate the efficiency of the search algorithm, the two-phase algorithm was implemented using two different search algorithms. In the first case a

Table 9. Performance comparison of the RT algorithm.

Evaluation type	Nodes explored	Time spent for exploration
RT algorithm (job shop 3 × 3)	1,232	5 s
CPN tools (job-shop 3 × 3)	1,232	3 s
RT algorithm (job-shop 6 × 6)	117,680	1.5 h
CPN tools (job shop 6 × 6)	117,680	3 h

pure sequential search was coded to analyse the stored nodes and detect if a node was an S-old node or if it was a new one. In the second case the binary search was implemented in the algorithm to make the same analysis. The benchmark problems were run with the two search algorithms and the results are those that are presented in Table 8.

The column *S-old nodes found* refers to the S-old nodes that must be analysed by the second phase of the algorithm in order to decide which sub-tree produces better time values. The *performed searches* column refers to the total number of searches performed by the algorithm to generate and evaluate the S-old nodes. The time consumption column refers to the total time that takes the algorithm to give a proper solution. It is clear the improvement was obtained with the proposed implementations in the search algorithm, since in most of the cases the obtained reduction averages 80%.

3.4.3. Overall performance evaluation

The evaluation of the algorithm consisted of comparing the time spent by the algorithm to generate and analyse the SS and compare the obtained results with an available tool that performs the same type of analysis. For this purpose, two models were coded (J-S 3 × 3 and J-S 6 × 6) and the results were compared against the CPN Tools software ([www.daimi.au.dk/~cpntools/](http://www.daimi.au.dk/~cpntools/)). The obtained results are presented in Table 9. This test gives insight about the efficiency of the transition-evaluation algorithm together with the implemented search techniques and analysis approach.

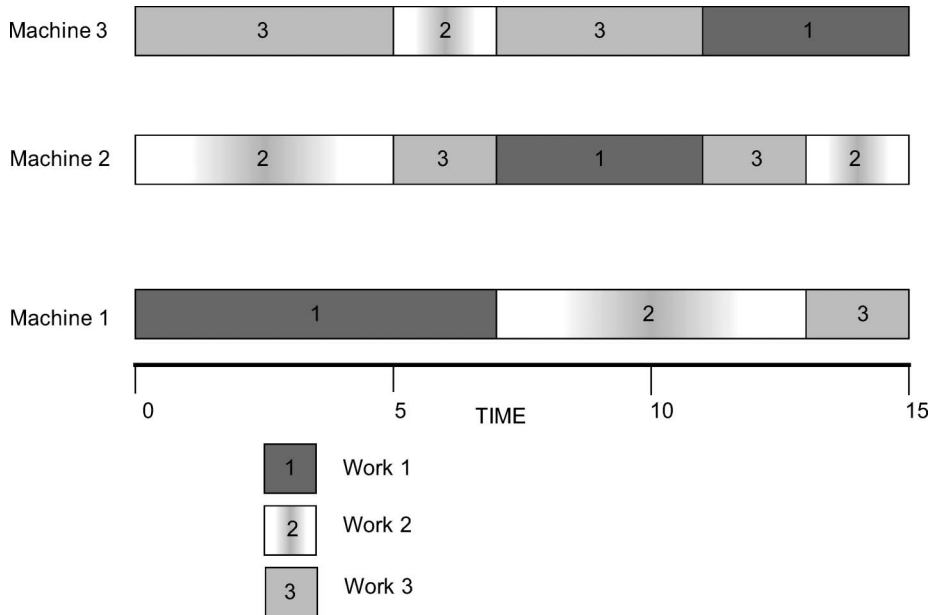


Figure 14. Gantt diagram of the optimal scheduling for the 3 × 3 job-shop.

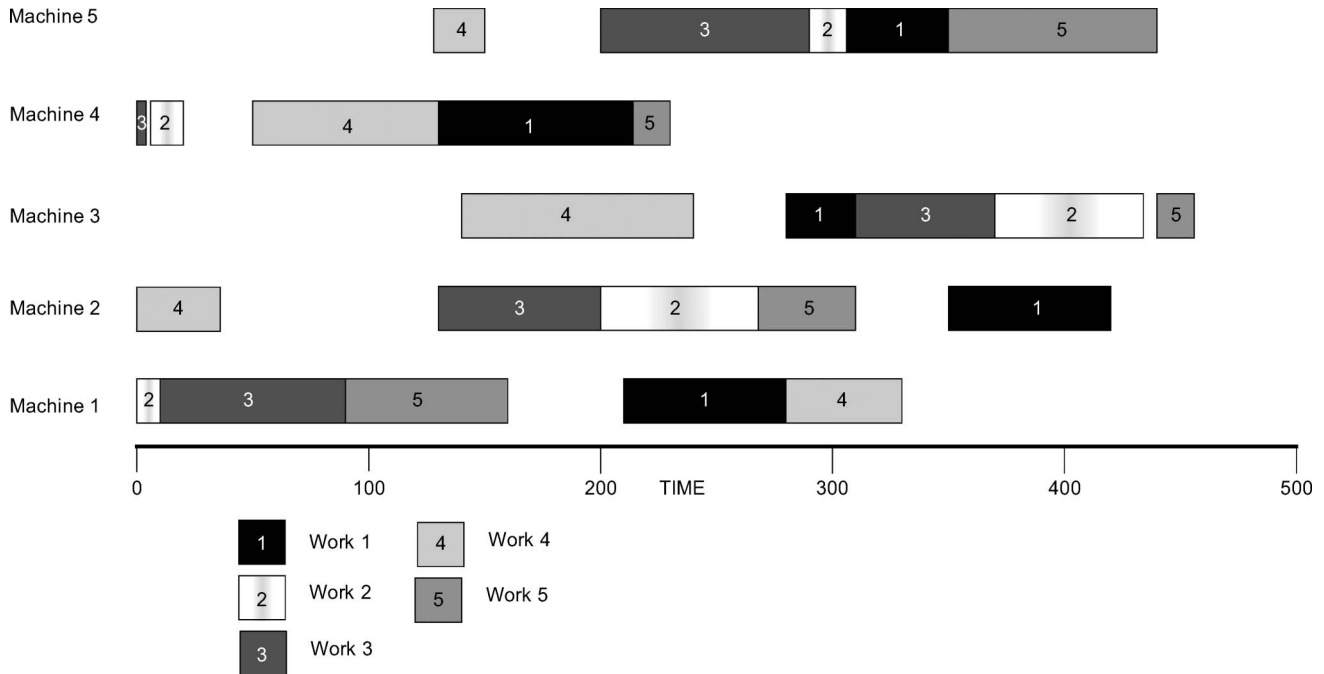


Figure 15. Gantt diagram of the optimal scheduling for the 5 × 5 job-shop.

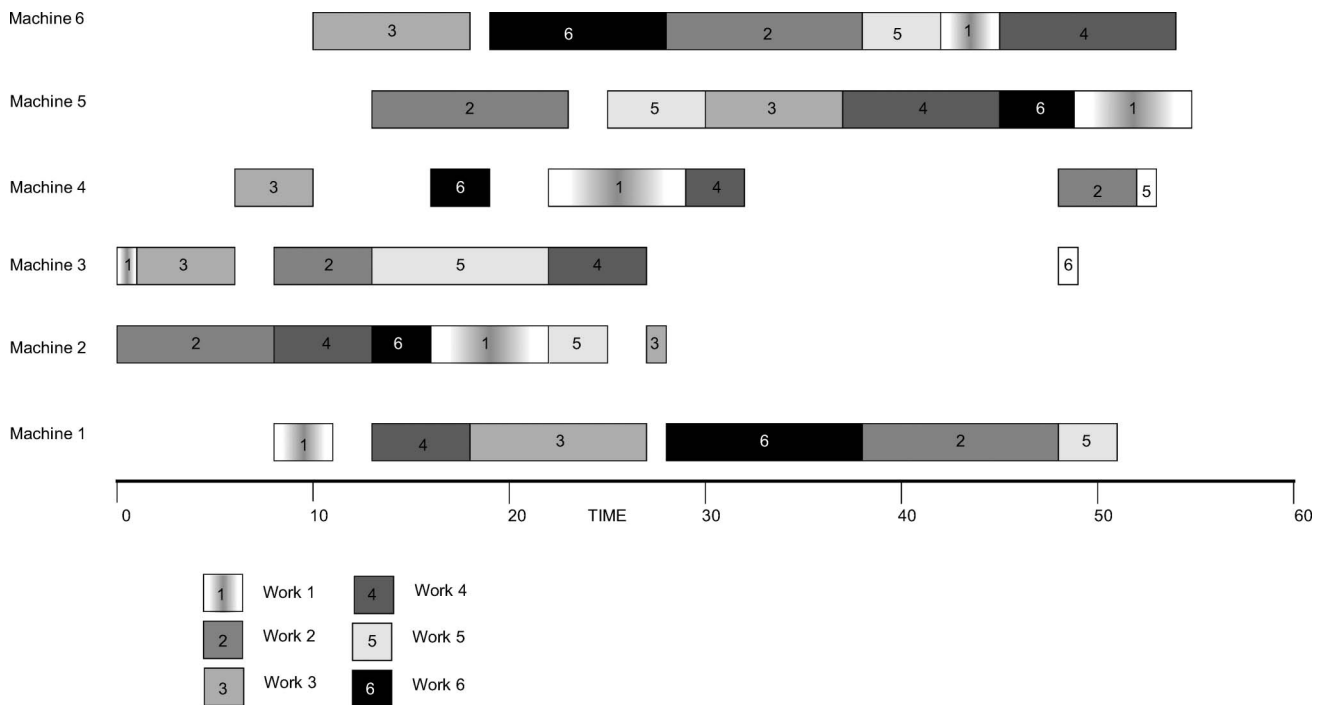


Figure 16. Gantt diagram of the optimal scheduling for the 6 × 6 job-shop.

It can be appreciated from the table that with small SSs, the algorithm does not perform as efficiently as with CPN Tools, but with a greater number of nodes, the performance is considerably better. The reason of

this result might lie in the fact that CPN Tools perform an exhaustive search while the two-step approach avoids the exhaustive search through the analysis of the S-old nodes.

The feasible paths that can be obtained with the proposed approach can be easily drawn in a Gantt graph to visually evaluate the proposed scheduled for these different problems. In Figures 14–16 the schedulings obtained are presented as Gantt's charts. These schedulings correspond to the ones presented in literature as the optimised ones (Fang *et al.* 1993).

#### 4. Conclusions

In this article, the most important elements to implement a simulator/optimiser based on the exploration and analysis of the CPN's compact timed SS have been presented.

The implementations presented focus on how to make smart data storage, a transition evaluation technique based on constraint programming for CPN models and an efficient approach to detect the new or repeated markings which together with the time analysis algorithm (Mujica *et al.* 2010) are the main elements for developing an efficient simulator/optimiser based on the exploration of the state space of timed CPN models.

The proposed algorithms were implemented and several tests were performed in order to illustrate the performance improvement when analysing TCPN models. The tests performed in section 3.2 clearly show that managing the information in a compact way allows storage of a bigger SS than the one that could be stored using a classical approach (storing all the timed markings). Section 3.4 shows the improvements in results obtained for a typical academic problem such as the job-shop problems. In this section, the job-shop problems have been solved with the proposed algorithm and the results obtained were compared to available CPN software (CPN tools). The results clearly show the advantages of the proposed algorithms over the available tools when dealing with models that present big SSs.

The obtained results show that the approach is well suited for the goal of the state exploration under time constraints. The search techniques implemented provide challenging computational benchmarking results when applied to industrial scheduling problems with a considerable amount of states to be evaluated.

The evaluation algorithm can be further improved if some heuristics are implemented for solving problems under particular objectives (logistic, manufacture, etc.). Therefore, this approach is proposed as adequate for dealing with industrial optimisation problems, which can be also used as the base for optimisation tools.

#### Acknowledgements

The authors would like to thank the Spanish Ministry of Education for the project CyCIT (TRA2008-05266/TAIR) and the AGAUR agency for the project AGAUR (2009 SGR 629) for supporting this research.

#### References

- Adballah, I.B., ElMaraghy, H., and ElMekkawy, T., 1998. An efficient search algorithm for deadlock free scheduling in FMS using Petri nets. *IEEE International Conference on Robotics Automation*, 2, 1793–1798.
- Chan, F.T.S., *et al.*, 2010. Operation allocation in automated manufacturing system using GA-based approach with multifidelity models. *Robotics and Computer-Integrated Manufacturing*, 26 (5), 526–534.
- CPN Tools, Daimi, Aarhus University, Available from: <http://cpntools.org/> [Accessed 21 December 2010].
- Dashora, Y., *et al.*, 2007. Deadlock-free scheduling of an automated manufacturing system using an enhanced coloured time resource Petri-net model-based evolutionary endosymbiotic learning automata approach. *International Journal of Flexible Manufacturing Systems*, 19 (4), 486–515.
- Dauzère-Peres, S. and Lasserre, J.B., 1994. *An integrated approach in production planning and scheduling. Lecture notes in economic and mathematical systems*, vol. 411. Berlin: Springer-Verlag.
- Fang, H.L., Ross, P., and Corne, D., 1993. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In: *Proceedings of the fifth international conference on genetic algorithms*. San Mateo, CA: Morgan Kaufman, 375–382.
- Gaeta, R., 1996. Efficient discrete-event simulation of coloured Petri nets. *Transactions on software Engineering*, 22 (9), 629–639.
- Gonnet, G.H., 1984. *Handbook of algorithms and data structures*. London: Addison-Wesley.
- Gradišar, D. and Mušič, G., 2007. Production-process modelling based on production-management data: a Petri-net approach. *International Journal of Computer Integrated Manufacturing*, 20 (8), 794–810.
- Jensen, K., 1997a. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Vol. 1. Berlin: Springer-Verlag.
- Jensen, K., 1997b. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Vol. 2. Berlin: Springer-Verlag.
- Jensen, K. and Kristensen, L.M., 2009. *Coloured Petri nets: modeling and validation of concurrent systems*. Berlin: Springer-Verlag.
- Kumar, R.R., Singh, A., and Tiwari, M.K., 2004. A fuzzy based algorithm to solve the machine-loading problem of a FMS and its neuro fuzzy petri net model. *The International Journal of Advanced Manufacturing Technology*, 23 (5), 318–341.
- Lalas, C., *et al.*, 2006. A simulation-based hybrid backwards scheduling framework for manufacturing systems. *International Journal of Computer Integrated Manufacturing*, 19 (8), 762–774.
- Lee, D.Y. and DiCesare, F., 1994. Scheduling flexible manufacturing systems using Petri nets and heuristic search. *Transactions of Robotics and Automation*, 10 (2), 123–132.
- Liu, C.M. and Wu, F.C., 1993. Using Petri nets to solve FMS problems. *International Journal of Computer Integrated Manufacturing*, 6 (3), 175–185.

- Mortensen, K.H., 2001. Efficient data structures and algorithms for a coloured Petri nets simulator. In: *Proceedings of CPN workshop*. Aarhus, Denmark: University of Aarhus, 57–74.
- Mujica, M., Piera, M.A., and Narciso, M., 2010. Revisiting state space exploration of timed coloured petri net models to optimize manufacturing system's performance. *Simulation Modelling Practice and Theory*, 18 (9), 1225–1241.
- Narciso, M., Piera, M.A., and Guasch, A., 2009. A Methodology for solving logistic optimization problems through simulation. *Simulation: Transactions of the Society for Modeling and Simulation International*, 86 (5–6), 369–389.
- Renna, P., 2010. Job shop scheduling by pheromone approach in a dynamic environment. *International Journal of Computer Integrated Manufacturing*, 23 (5), 412–424.
- Reyes, A., *et al.*, 2002. Integrating Petri Nets and hybrid heuristic search for the scheduling of FMS. *Computers in Industry*, 47 (1), 123–138.
- Störrle, H., 1998. *An evaluation of high-end tools for Petri nets*. Munich: LMU.
- Tchako, J.F.N., *et al.*, 1994. Modelling with coloured Petri nets and simulation of a dynamic and distributed management system for a manufacturing cell. *International Journal of Computer Integrated Manufacturing*, 7 (6), 323–339.
- Tsang, E., 1993. *Foundations of constraint satisfaction*. London: Academic Press.
- Valmari, A., 1998. The state explosion problem. *Lecture Notes in Computer Science*. Vol. 1491. London: Springer-Verlag, 429–528.
- Van der Aalst, W.M.P., 1995. *Petri net based scheduling*. Computing Science Reports, 95/23. Eindhoven University of Technology. Available from: [www.win.tue.nl/win/cs](http://www.win.tue.nl/win/cs) [Accessed 27 December 2010].
- Wells, L., 2002. Performance analysis using coloured Petri nets. In: *Proceedings of the 10th IEEE international symposium on modeling, analysis, and simulation of computer and telecommunications systems (MAS COTS'02)*. Fort Worth, Texas, New York: IEEE, 0217.
- Yogeswaran, M., Ponnambalam, S.G., and Tiwari, M.K., 2009. An efficient hybrid evolutionary heuristic using genetic algorithm and simulated annealing algorithm to solve machine loading problem in FMS. *International Journal of Production Research*, 47 (19), 5421–5448.
- Yu, H., *et al.*, 2003. Combined Petri net modelling and AI based heuristic hybrid search for flexible manufacturing systems – Part 1. Petri net modelling and heuristic search. *Computers and Industrial Engineering*, 44 (4), 527–543.