International Journal of Production Research
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

An improved time line search algorithm for manufacturing decision-making

Miguel Mujica Mota & Miquel Angel Piera
[a] Department of Telecommunications and Systems Engineering , Universitat Autonoma de Barcelona , Barcelona , Spain

Published online: 04 Oct 2013.

To cite this article: Miguel Mujica Mota & Miquel Angel Piera , International Journal of Production Research (2013): An improved time line search algorithm for manufacturing decision-making, International Journal of Production Research, DOI: 10.1080/00207543.2013.839892

To link to this article: http://dx.doi.org/10.1080/00207543.2013.839892

International Journal of Production Research, 2013
http://dx.doi.org/10.1080/00207543.2013.839892

An improved time line search algorithm for manufacturing decision-making

Miguel Mujica Mota* and Miquel Angel Piera

Department of Telecommunications and Systems Engineering, Universitat Autonoma de Barcelona, Barcelona, Spain

(*Received 20 December 2012; accepted 26 June 2013*)

The coloured Petri net formalism has been recently used to analyse and optimise manufacturing systems making use of the state space (SS) analysis. This approach has great potential for scheduling and production planning purposes when it is properly implemented. In this article, an improved version of the algorithm known as the *time line search* for optimising the makespan of manufacturing models is presented. The algorithm has been developed for the use in a compact SS of coloured Petri net models in order to analyse the highest possible number of manufacturing configurations for the improvement of the makespan of a production system. The proposed algorithm can be used for the developing of decision support tools in manufacturing or operational decision-making.

Keywords: timed coloured Petri nets; state space; manufacturing; optimisation

1. Introduction

The management of resources in manufacturing industries has been traditionally a challenging problem due to the multiple configurations that can be achieved depending on the number of resources, jobs and constraints that need to be coordinated in order to produce a particular good. Depending on the type of constraints and precedence sequences, some problems can be modelled as job shops, flow shops or open shops (Pinedo 1995). Moreover, due to the importance of these types of problems, they have been faced by scientific community using techniques that range from traditional mathematical formulations (Pinedo 2005; Ravindra and Mathirajan 2011) to heuristics (Ghoul et al. 2007; Pitts and Ventura 2009) in combination with modelling formalisms such as Petri nets (Huang, Sun, and Sun 2008; Mejia and Montoya 2009). Depending on the objective of the study, different performance indicators can be used to measure the proper coordination of resources in a particular scenario. One important performance indicator in manufacture systems (which has its counter parts in other systems) is the *makespan* which quantifies the total time that takes a process for being completed. Therefore, in order to evaluate the advantage of one configuration over another, it is necessary to compare the obtained makespan values. The evaluation of system scenarios to determine the best decision variable values by means of simulation is an accepted approach due to the ability to support several description levels 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 (Merkuryev et al. 2009).

In recent years, the use of causal modelling formalisms to perform optimisation tasks has acquired good acceptance in the scientific community since it is possible to develop models with a high accuracy level supporting the analysis of emergent dynamics. In particular, the use of Petri nets in its different variations allows taking advantage of the analysis tool known as the reachability graph (Jensen, Mailund, and Kristensen 2001; Mejia and Montoya 2009). With the use of this graph it is theoretically possible to evaluate all the possible configurations that can be reached by a certain model starting from an initial one, therefore an optimisation problem can be transformed into a search one, but in practical cases the latter cannot be achieved due to memory limitations. The analysis of the reachability graph has given good results with heuristics for models developed in ordinary Petri nets (Lee and DiCesare 1994; Mejia and Montoya 2008; Xiong and Zhou 1998). On the other hand when the models are developed using the coloured or timed formalisms, the information of the modelled systems can be encoded exploiting the particular characteristics of the formalisms (colours, expressions and time stamps) resulting in more abstract and complex models than the ones that could be developed with the ordinary Petri nets; thus, preventing a straightforward way of implementing the analysis of the reachability graph. In order to cope with the combinatorial explosion (Valmari 1998), the scientific community faces the challenge of developing techniques for the efficient exploration of the graph. This is the reason why efforts have been devoted to

*Corresponding author. Email: miguelantonio.mujica@uab.cat develop approaches that aim at avoiding the memory saturation such as the comeback method (Westergaard et al. 2007) or condensed versions of the state space (SS) (Jensen 1996). Moreover, heuristic approaches are being developed for exploring the SS of timed coloured Petri nets (TCPN) to obtain quasi-optimal solutions in short times (Mujica, Piera, and Narciso 2010; Zuniga, Piera, and Narciso 2011). All these approaches aim at alleviating as much as possible the memory consumption to maintain all the information of the graph.

The authors have developed an algorithm for the generation and exploration of the reachability graph called *time line search algorithm* (TLS) for manufacturing models developed in TCPN (Mujica and Piera 2010). This algorithm generates and analyses the SS in two phases. The implementation was conceived based on the idea of the *sweep line method* (SLM) (Kristensen and Mailund 2002) but using a compact version of the timed SS. The performance of the implementation yielded very good results with models whose SSs were small enough to be stored in the computer memory.

The original algorithm (that will be called in the future as the TLS VerA-algorithm) presented limitations in the evaluation processes that caused some time penalties in the whole performance. The new implementations are devised with the purpose of overcoming those drawbacks giving as a result a better performance of the algorithm. The novel implementations have been tested with two problems, first with the industrial model of an eye-glass CNC machine (Mujica and Piera 2009) and second with the 6×6 job shop (Dauzére-Peres and Lasserre 1994), which is a well-known academic problem.

The rest of the paper is organised as follows. Section 2 introduces the TCPN modelling formalism. Section 3 describes the compact timed SS and addresses its advantages for performing optimisation tasks. Section 4 summarises the two-phase optimisation algorithm. Section 5 illustrates the recent implementations in the current algorithm and provides the correspondent pseudo codes for implementing the main procedures behind the algorithm. Also, in this section, the experimental results are presented and finally Section 6 concludes the paper and discusses future implementations in the algorithm.

2. Timed coloured Petri nets

CPN is a simple yet powerful modelling formalism that allows to properly model discrete-event dynamic systems which present concurrent, asynchronous or parallel behaviour (Jensen 1997; Moore and Gupta 1996). CPN belong to the so-called *high level Petri Nets* because they are characterised by a combination of Petri nets and programming languages (Jensen and Kristensen 2009). Therefore, CPN allows the construction of compact and parameterised models. Moreover in CPN, the tokens have attributes (known as colours) that allow keeping track of the information flow which is an important characteristic for decision-making. Furthermore, in order to evaluate systems performance, it is necessary to extend the formalism attaching time stamps to the tokens representing the time at which the tokens are available (i.e. the time at which they can be removed by an occurring transition), a global clock representing model time and a time delay to transitions representing the time that will be consumed by the activity. A time delay inscribed on a transition applies to all output tokens created by that transition. The time stamps of tokens and delays in transitions are written after the symbol @ (e.g. $1'(3,5)@4$ represents that this token is available at global time 4).

CPN makes necessary the use of expressions that contain variables attached to the connecting arcs, and to define the corresponding constraints associated to transitions (*Guards*). Therefore, in the formal definition, *EXPR* denotes the expressions provided by the inscription language (e.g. CPNML in the case of CPN Tools or C++ expressions in the case of PetriSimm), 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 free variables in an expression *e* is denoted *VAR*[*e*] and the type of a variable *v* is denoted *TYPE*[*v*]. A free variable is a variable which is not bound in the local environment of the expression (i.e. has no value assigned to it).

Formally the timed version of CPN used in this work can be defined as follows.

Definition 1. TCPN.

TCPN = (P, T, A, $\sum$, 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) $\sum$ is a finite set of non-empty colour sets with time values associated to them.
(5) V is a finite set of typed variables such that *Type* $[\upsilon] \in \sum$ for all variables $\upsilon \in V$.
(6) C: $P \to \sum$ is a colour set function assigning a timed colour set to each place.

(7) G: $T \rightarrow EXPR$ is an expression known as *guard* assigning an expression to each transition T such that *Type* $[G(t)]$ = *Boolean* for all $t \in T$.
(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 (graphically represented with a '@' sign) that assigns a delay, expressed in integer values, to each transition such that *Type* $[D(t)] \in N$ for all $t \in T$.
(10) I: $P \rightarrow EXPR$ is an initialisation function assigning an initial timed marking to each place p such that *Type* $[I(p)] = C(p)$.

In TCPN context, the state of every model is defined by the representation of the coloured tokens with their time stamps for each place p of the model, this representation is also called the *timed marking.*

Definition 2. 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* since the expressions contain also the time stamps.

Definition 3. 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 $M^U(p) \in C(p)$ representing the untimed marking of place p. In this case, the expressions do not contain any time information.

When a transition occurs, the output tokens will have a time stamp Δt time units larger than the current global clock Gc (formula (1)). This new time stamp simulates time delay due to the execution of an activity and also 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 Δt time units.

$$t_o = Gc + \Delta t \tag{1}$$

where t_o is the time stamp 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 Δt is the time associated with the transition.

3. The compact timed SS

The reachability graph is a directed graph which has been traditionally used by scientific community for the verification and analysis of behavioural properties of timed and non-timed CPN models (Christensen et al. 2001; Kristensen and Mailund 2002; Wolf 2007). The reachability graph is also known as the SS of Petri and coloured Petri net models because it represents all the different reachable states from an initial one. The SS analysis can be performed with timed or untimed models to evaluate properties, such as liveness, boundedness and reachability among others (Jensen, Mailund, and Kristensen 2001) to determine the behaviour of the modelled system.

Due to the state explosion problem (Valmari 1998), some authors have devised different ways of representing the SS (Chiola et al. 1997; Jensen, Mailund, and Kristensen 2001) in order to have better analysis capabilities with the available resources. For all the representations, the so-called *old node* plays an important role in order to avoid memory saturation since it represents identical states of the model which can be reached through different processes (transitions firings).

Definition 4. Let $\mathfrak{M}^T$ be the set of timed markings of a SS, and $M_i^T \in \mathfrak{M}^T$, $M_k^T \in \mathfrak{M}^T$ be timed markings. A state M_k^T will be called *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, i.e. $M_k^T = M_i^T$.

It is possible to reduce the computational effort to analyse the different states of the tree when the symmetry is exploited in such a way that nodes with the same underlying non-timed marking (e.g. for a marking $[1'(2,4)@3,\ 1'(3,5)@5 + 1(3,4)@4]$, the non-timed marking would be $[1'(2,4), 1'(3,5) + 1'(3,4)]$) are grouped in one node called S-old node.

The S-old nodes result also useful for developing a compact version of the SS for TCPN models. Thus, the symmetric old node or *S-old node* is defined as follows:

Definition 5. Let $\mathfrak{M}^T$ be 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^U and M_k^U.

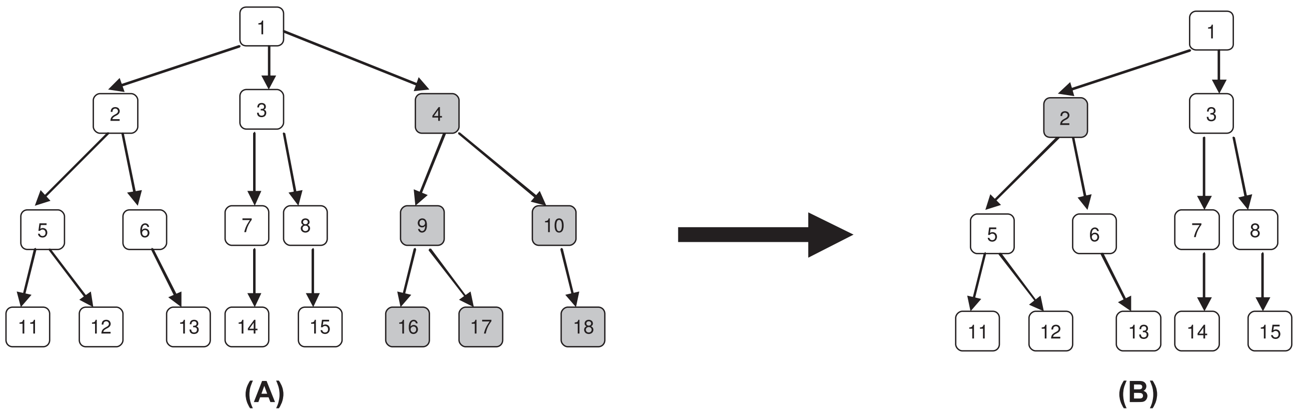


Figure 1. Use of S-old nodes.

A marking M_i^T is called a symmetric old node, S-*old node*, to another M_k^T marking when the following condition holds:

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

The SS representation achieved using the S-old nodes is called the *compact timed state space* (CTSS) which allows reducing the amount of physical space needed to store the information generated during the exploration of the SS.

Figure 1 is used to illustrate the representation that can be achieved using the S-old nodes. The feasible reduction is illustrated in Figure 1(B), where node #2 represents the S-old node stored with a reference to the time values of its S-old node (node #4). If the time information of the S-old node #4 is needed to evaluate the potential of its branch to improve the values of the objective node, it can be performed using the time information stored along with the S-old node #2 and the information of the subtree that hangs from it.

4. The two-phase algorithm

The use of the CTSS for optimising the makespan of industrial systems was originally devised through an algorithm in two phases (Mujica, Piera, and Narciso 2010). The search used in this algorithm has been performed using a *depth-first search* (DFS) where one starts at the root node and explores as far as possible each branch before performing a backtrack (Cormen, Leiserson, and Rivest 1990). In order to make the paper self-contained, the key aspects of the algorithm are presented in this section.

4.1 *Verifying and optimising industrial systems in two phases*

The first phase consists in the generation of the different states of the model under a DFS basis until the objective marking is found. The second phase performs the time analysis of the S-old nodes in order to optimise the found path. The advantages of this approach over similar ones (Berthomieu, Ribet, and Vernadat 2004; Jensen, Mailund, and Kristensen 2001) are deadlock free paths and some behavioural properties of the model (such as deadlocks, reachable states, etc.) can be verified during the first phase allowing stopping the process at the end of this procedure. In addition, the S-old nodes can be analysed in the second phase using a cost function in order to perform an optimisation based on a specific criterion. In the work presented here, the cost function to be minimised is the makespan.

4.1.1 *SS generation phase*

In the first phase, a CTSS is generated. The algorithm generates as many new states as different token combinations enable the transitions based upon 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 synchronises with the earliest firing time. During the generation phase, the search of previously generated S-old nodes is also performed.

In this algorithm, a *depth-first search* is used for exploring the tree. For this purpose, two lists are implemented.

(1) 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).
(2) ON-list: is used for storing the S-old nodes and time stamp information. The time values of the repeated S-old nodes found during the generation phase are stored in data blocks (time stamp list) related to the correspondent S-old node.

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 for later analysis as it is illustrated in Figure 2.

This node management approach 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 only once during the first phase. 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 for the model.

4.1.2 *Second phase: optimisation of time values*

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

The second phase analyses the time values of the S-old nodes following a simple sequence on the ON-list (from the first to the last element). It takes an S-old node from the ON-list and compares its time stamp values to the ones from the S-old node used in the generation step (left column in Figure 2(B)). The comparison is performed applying the empirical formulas presented by Mujica, Piera, and Narciso (2010).

Figure 3 illustrates the analysis performed. Given any two S-old nodes M_i and M_J, the time stamp lists are compared on a one-to-one basis and three possible outcomes are obtained after the comparison:

- 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 time stamp values of the M_i state are smaller than the ones of M_J state. In this case, nothing is updated since the sub branches would have better values than those that can be obtained from M_J.

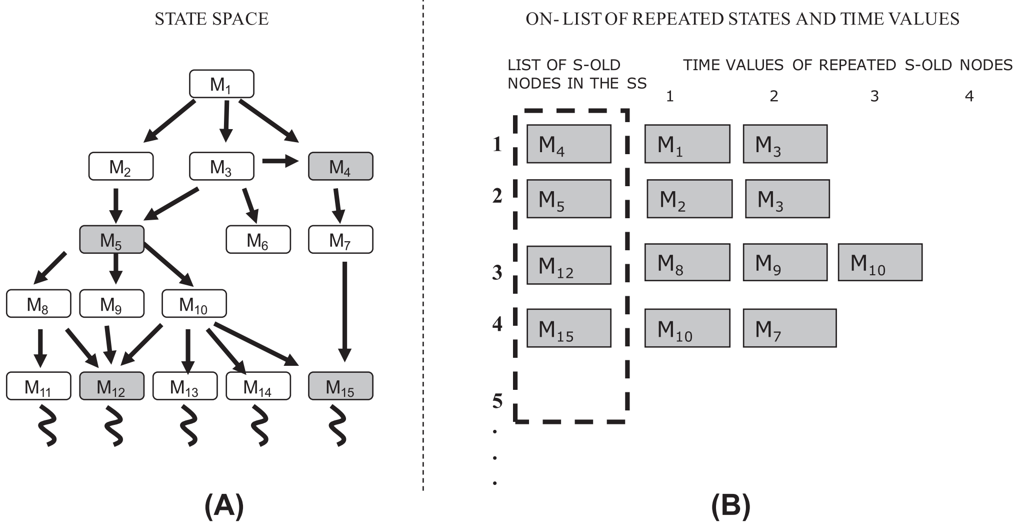

STATE SPACE
M_1
M_2
M_3
M_4
M_5
M_6
M_7
M_8
M_9
M_{10}
M_{11}
M_{12}
M_{13}
M_{14}
M_{15}
(A)
ON- LIST OF REPEATED STATES AND TIME VALUES
LIST OF S-OLD NODES IN THE SS
TIME VALUES OF REPEATED S-OLD NODES
1 2 3 4
1 M_4 M_1 M_3
2 M_5 M_2 M_3
3 M_{12} M_8 M_9 M_{10}
4 M_{15} M_{10} M_7
5
(B)

Figure 2. Lists to store the generated SS. Reprinted from Mujica, Piera, and Narciso 2010, with permission from Elsevier.

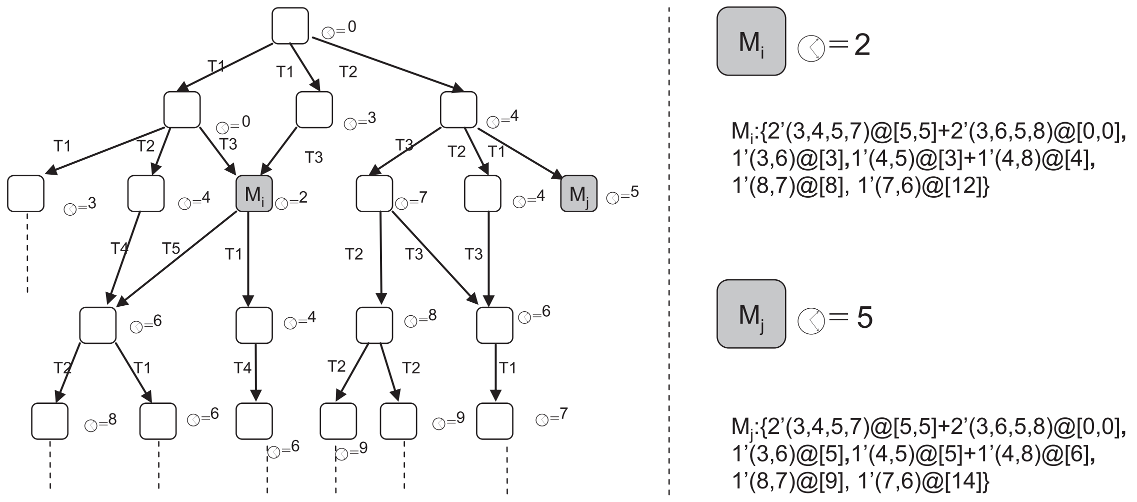

T1
T2
T3
T4
T5
M_i
M_j
=0
=2
=3
=4
=5
=6
=7
=8
=9
M_i:{2'(3,4,5,7)@[5,5]+2'(3,6,5,8)@[0,0], 1'(3,6)@[3],1'(4,5)@[3]+1'(4,8)@[4], 1'(8,7)@[8], 1'(7,6)@[12]}
M_j:{2'(3,4,5,7)@[5,5]+2'(3,6,5,8)@[0,0], 1'(3,6)@[5],1'(4,5)@[5]+1'(4,8)@[6], 1'(8,7)@[9], 1'(7,6)@[14]}

Figure 3. Time analysis of the repeated states.

- 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, an exploration through the best branch is performed until the objective marking is reached and then a comparison of the values for the objective state can be made.

The feasible path is updated with the new time values once it is detected that the time values minimise the makespan; this procedure is performed maintaining time consistency in the updated path.

The details of the algorithm can be consulted in Mujica, Piera, and Narciso (2010) and those related to the implementations in Mujica and Piera (2011).

5. New TLS algorithm

In this section, recent implementations that outperform the two-phase algorithm are presented.

5.1 *The TLS VerA-algorithm*

As it has been mentioned, the idea behind this search algorithm is the notion of time progress. This idea was taken from the algorithm known as the SLM (Kristensen and Mailund 2002) that introduces the idea of evaluating a SS based on a notion of progress instead of a depth-first search or a simple breath-first search.

This idea was taken for the exploration of the CTSS; the earliest firing time of every marking is used as a key that matches the value of the incremental variable. The nodes that share the same firing time will be grouped and evaluated in order of appearance until all nodes of the group have been evaluated.

The main elements needed for the TLS VerA-algorithm are:

- A key $\mathbf{K_i}$ that is assigned to each node in the CTSS. This key will be used to determine the sequence of evaluation.
- An incremental variable (i.e. global clock) **Gc** that determines the group of nodes that will be evaluated afterwards. Those nodes whose keys match the value of **Gc** will be the ones to be evaluated next.
- An ordered SS-list with the sequence of evaluation is based on the progress measure.
- An ordered ON-list (initially empty).

The key $\mathbf{K_i}$ is calculated from the time values of the markings using formula (2). It uses the list of time stamps of the tokens present in the marking and the global clock of the marking:

$$\begin{aligned} tline_k : \quad & \mathbb{N}^k \times \mathbb{N} \to \mathbb{N} \\ & (T_i, Gc) \mapsto \mathrm{Max}\{\mathrm{Min}\{T_{i1}, \ldots, T_{ik}\}, Gc\} \end{aligned} \tag{2}$$

where $M_i^T \in \mathfrak{M}^T$ is a timed marking, T_i its correspondent time stamp list and Gc the global clock of the timed marking.

This formula is used to develop the ordered SS-list for the evaluation sequence. Based upon this procedure, there will be different sets of nodes that will differ from each other in the key value and the evaluation sequence will depend only on the value of the time variable.

5.2 *Limitations and improvements to the TLS algorithm*

Making use of the elements in the TLS VerA-algorithm, the SS is generated and analysed in two phases. The SS of the model in Figure 4 is used to illustrate the principles of the TLS algorithm.

A limitation appeared when the evaluation of some S-old nodes with good potential was delayed even if they could be fired with a global time earlier than the already generated node. Figure 5 illustrates this situation. The axis called *time variable* represents the values taken by the global clock while the nodes that fall behind the dashed lines represent those nodes whose firing time values are the same.

Under this approach, the sequence of group evaluation depends on the value of the global clock while the nodes within a group are being evaluated in order of appearance. The first group of nodes (nodes #1, #2, #3, #4, #5, #6 and #8) has been evaluated when the global clock had the value 0. If we put focus on node #6, during its evaluation, it generated nodes #12 and #13. In the case of node #12, the firing time was 6 time units (that is why it falls in the 6-time group) while the node #13 was 1 time unit. In the case of node #2, it generated node #7 at a firing time of 1 unit. When all the nodes of the 0 time group were evaluated, the global clock (Time Var) headed to the next value (1 time unit). When the second group of nodes was evaluated, the first node, node #7, generated the S-old node #12 and the node #16. If focus is put on the S-old node #12, it can be noted that it had been already generated by node #6 during the previous evaluation (with a firing time of 6 units). However, if node #12 was generated from node #7, the correspondent firing time would have been 2 time units!

In order to follow the principle of the time line, node #12 should have been evaluated when the driving variable reached 2 time units instead of 6 time units. The first improvement to the new algorithm aims at overcoming this problem, and it is explained in the following subsection.

5.2.1 *Improvement A: consistency evaluation*

A procedure which analyses on the fly whether a node is better or not than another one previously generated (based on their firing time) has been developed in order to overcome the limitation discussed in the previous subsection.

The improved algorithm will use the same elements of the TLS VerA-algorithm but a step will be added to verify that the firing times of the correspondent nodes are the smallest ones. Figures 6–8 illustrate the new procedure performed by the algorithm in order to overcome the limitation.

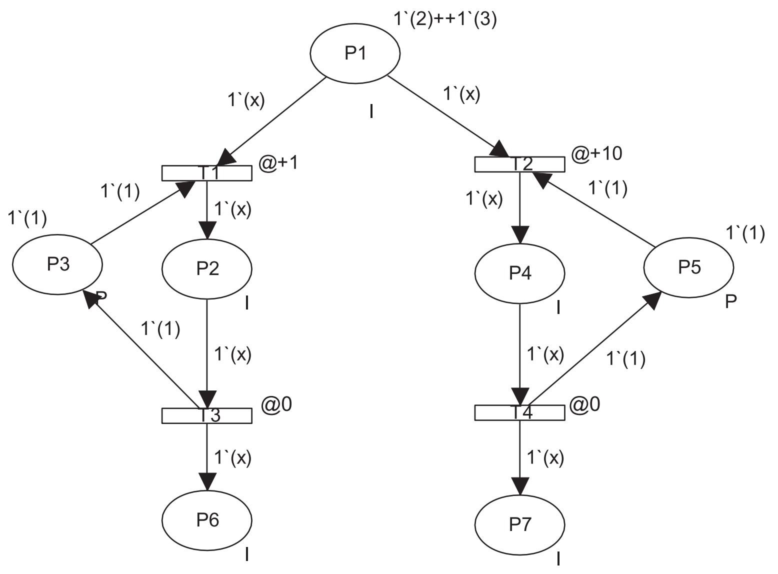


Figure 4. A shared-resource TCPN model.

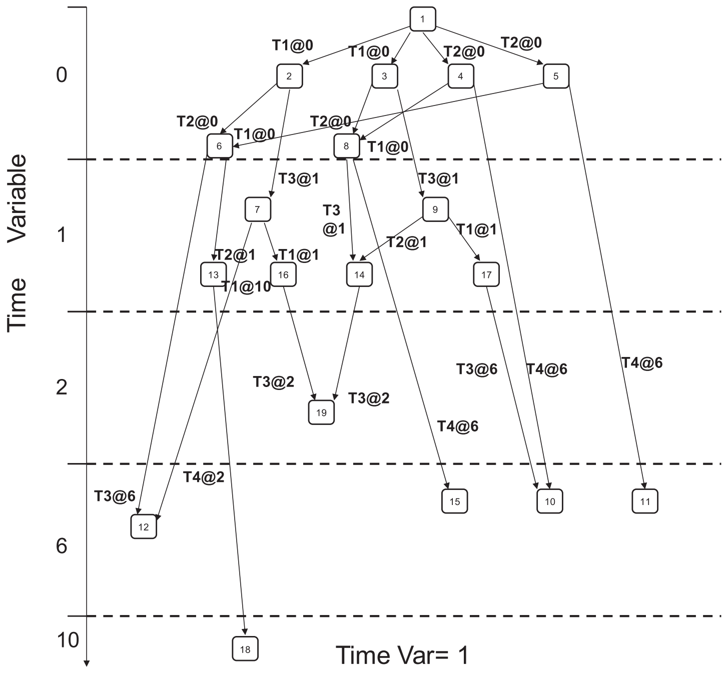


Figure 5. Generation of the CTSS using the TLS algorithm.

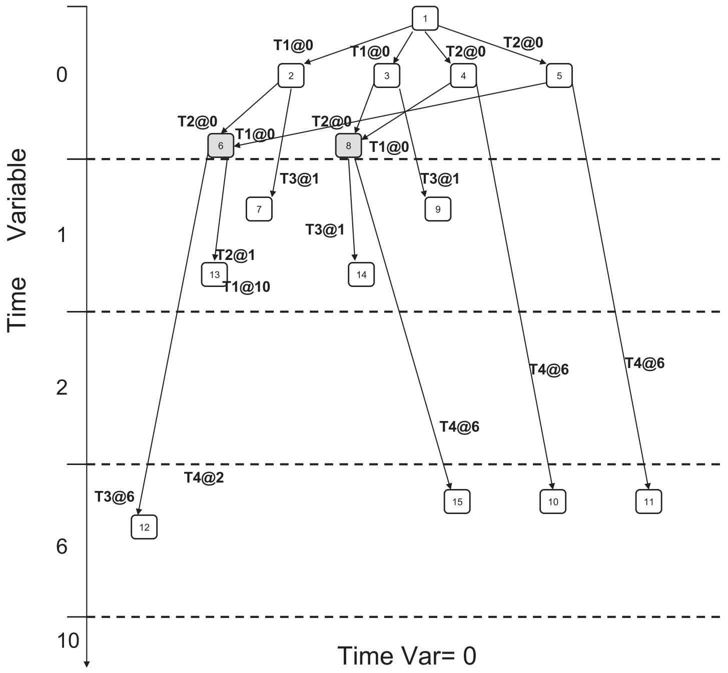


Figure 6. Evaluation of group with firing time = 0.

Step (*1*) Figure 6, during this instant of time, the first nodes to be evaluated are nodes #1, #2, #3, #4 and #5 which generate nodes #6 to #15. In these evaluations, nodes #6 and #8 are generated with a firing time of 0 units (T1@0, T2@0), therefore they fall within the first group. The time variable value will not change until all the nodes in the group of 0-firing-time value have been evaluated including nodes #6 and #8. The remaining nodes (with firing time greater than 0) take their place in the list until their time for evaluation approaches.

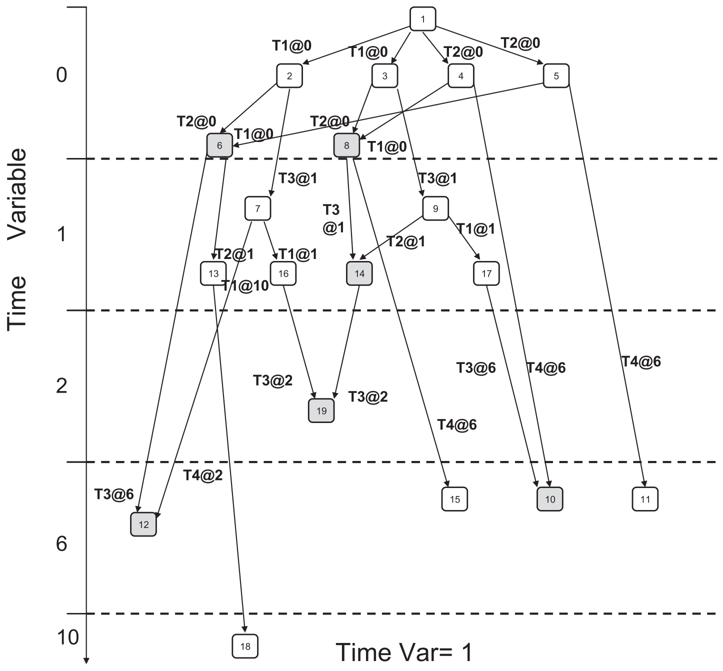


Figure 7. Time variable heading to firing time = 1.

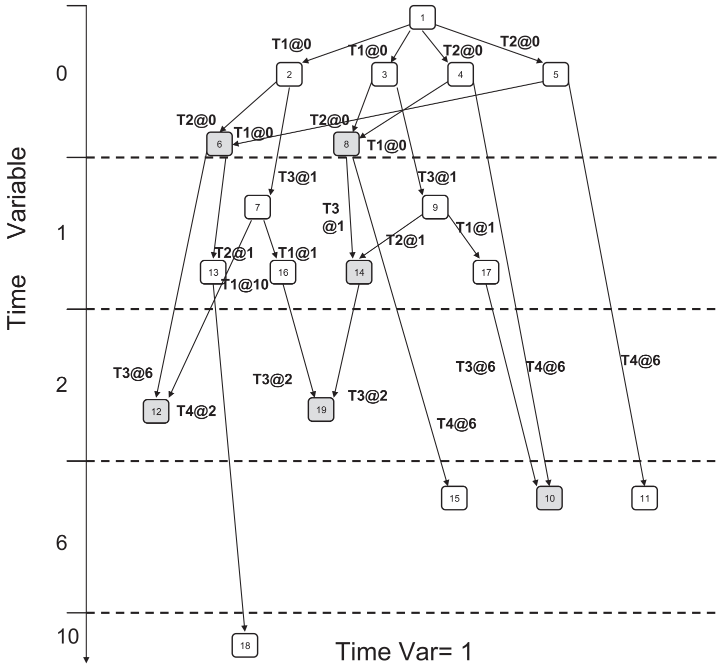


Figure 8. Updating the value of generated node.

Step (*2*) When all the nodes of the 0-firing-time value group have been evaluated, the S-old nodes generated so far (the grey-shaded nodes in the figure) must be analysed in order to determine if they have the smallest firing-time value for the successive evaluations. In Figure 6, the S-old nodes #6 and #8 are generated with the 0-firing-time value from any of their father nodes.

Step (*3*) Figure 7 illustrates the next step in the evaluation procedure, the global clock heads to the new value (1 time unit). The group that matches the new time value is evaluated following the sequence #7, #9, #13 and node #14. The resulting SS from that evaluation sequence is illustrated in Figure 7. It can be appreciated that four new S-old nodes have been generated during this evaluation namely nodes #10, #12, #14 and #19.

In this figure, node #12 represents the S-old node that has been generated earlier during the generation algorithm by the evaluation of node #6.

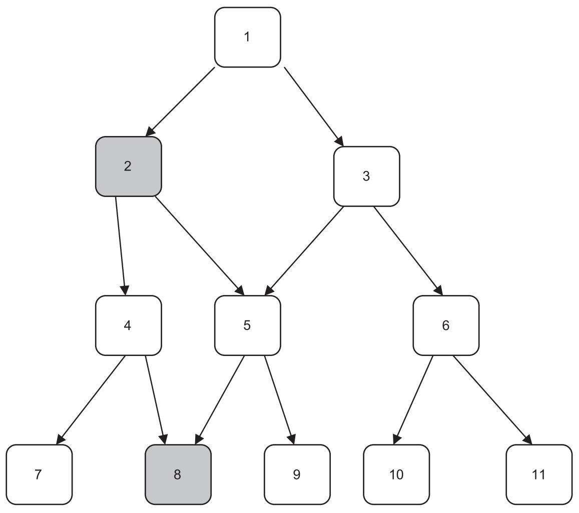
When all the nodes from the group of 1-firing-time value have been evaluated, only the new S-old nodes are verified in order to ensure that the firing-time value is the smallest possible:

- The algorithm takes the list of the S-old nodes and makes a comparison between the firing times of the nodes that generate an S-old node.
- If a firing time that is less than the original firing time is found, it will update the time values of the found node (time stamps and global time) and the information related to the father node that generated the original node will be replaced with the one that produced the best time values.
- If the node has not been evaluated yet, it will be removed from the group (in the SS-list) that originally belonged to and it will take its position in the new group (Figure 8; one group with smaller firing time than the original, note node #12).
- If the node has been already processed, then it will update the time values and afterwards it will update the time values of the branch that hangs from the node.

5.2.2 *Improvement B: differentiating similar markings*

Another limitation arises when two nodes are fired at the same global time and the operations take similar time; in that case the resulting markings might have different colour configuration but the time stamps of both markings will have similar values. This situation prevents a clear decision on which father node information must be maintained for the subsequent evaluations. In order to establish a difference between the potential outcomes of both nodes, the following decision rules have been implemented:

- If two states have the same firing time, the one with the earliest time stamp will be selected. This action assumes that the selected token will be ready earlier than the one from the other state.



$\boxed{4}$:T1@3 [2'(3,4)@3,5 , 1'(1)@4 + 1'(2)@5 , 0 , 0]

$\boxed{5}$: T3@3 [2'(3,4)@3,6 , 1'(1)@5 + 1'(2)@6 , 0 , 0]

Figure 9. Differentiating two markings.

- If the firing time and the earliest time stamp of both nodes are the same then it will be selected the second earliest time stamp and so on until it is found one time stamp that differentiate both markings.

The previous implementation assumes that all tokens that compose the markings have the same probability of enabling a transition of the model. Figure 9 illustrates the developed procedure to distinguish between two nodes.

In this example, the S-old node #8 can be generated from nodes #4 and #5. Both nodes have the same firing time (3 time units), therefore they can only be differentiated evaluating their time stamps. The time stamp values that are taken into account for the decision have the values 3, 4 and 5 for the S-old node that can be generated from node #4 and 3, 5 and 6 for the S-old node that can be generated from node #5. The time stamp that differentiates both nodes is the second one; hence node #8 would be evaluated using the time values generated from node #4. If the S-old node # 8 was generated from another node different than #4, then the time stamps of node #8 would be updated with the new time values. If the node has not been evaluated yet, then the time-updating process would end in that node; if there was a sub branch that hanged from the node, then the time values of the complete branch must have been updated.

5.3 *Algorithm of the New TLS algorithm*

The algorithm of the new time line search is presented in the following pseudo code.

Algorithm 1. The New TLS algorithm

```
BEGIN
SET Gl_CLK := 0
SET X := 0
INITIAL_CHILDREN (initial_Marking, SS-list, ON-list)
DO
X:=X + 1
IF GL_CLK = KEY(SS-list(X)) THEN
   GENERATE_CHILDREN (SS-list(X), SS-list, ON-list)
ELSE
   GL_CLK := KEY(SS-list(X))
   TIMELINE_CONSISTENCY (ON-list, GL_CLK)
ENDIF
IF Stop_Criteria THEN EXIT LOOP
LOOP Until SS-list(X) = NULL
S_OldNode_EVALUATION (ON-list)
Generate_RESULTS
END
```

In this algorithm, the procedure INITIAL_CHILDREN evaluates the initial marking and sets up the SS-list and ON-list with the generated children nodes. There are two key procedures, namely GENERATE_CHILDREN and TIMELINE_CONSISTENCY. The former processes the elements in the SS-list when the KEY is equal to the driving variable GL_CLK, the latter verifies in the ON-list that the corresponding key of the markings is the one with the smallest possible value. In the GENERATE_CHILDREN procedure, the new markings are ordered in the SS-list in accordance to the value calculated by function (2) and subsequently this value will be used as the key for their sequence of evaluation. Because of the principle of the time line these new markings will take their place ahead of the marking under evaluation; in the worst case the new markings would fall within the same group of markings of the node under evaluation but they must be added to the SS-list in a position ahead of the pointer that aims at the current element. Once a new element is generated, it is verified whether it is an S-old node, if so it will be added to the ordered ON-list based on the value obtained by function (2).

The following pseudo code presents the procedure GENERATE_CHILDREN

Algorithm 2. Pseudo code of GENERATE_CHILDREN

```
Procedure GENERATE_CHILDREN (Marking1, SS-list, ON-list)
BEGIN
TRANS := 0
DO
New_Marking := EVALUATE_TRANSITIONS (marking1, TOT_TRANS)
IF Is_Old_Marking (New_Marking) = TRUE then
    TRANS := TRANS+1
    ADD New_MarkingTO_ORDERED ON-list
ELSE
    TRANS := TRANS+1
    ADD New_MarkingTO-ORDERED SS-list
END IF
REPEAT UNTIL TRANS = TOT_TRANS
END
```

The second key procedure is TIMELINE_CONSISTENCY; which makes an analysis of the keys within ON-list. It verifies that the key has the smallest possible firing time among the father nodes that point to that node. In case that it is found a better firing time among the father nodes that point to the node under evaluation, the key and the time elements of the S-old node are updated; if there is a sub branch that hangs from the S-old node, then the time values of the branch are updated as well.

The following pseudo code presents the procedure TIMELINE_CONSISTENCY.

Algorithm 3. TIMELINE_CONSISTENCY

```
Procedure TIMELINE _CONSISTENCY (ON-LIST1, CLOCK)
BEGIN
POS := Find_Position(ON-LIST1,CLOCK)
    FOR X = POS TOEnd Of ON-LIST1
VERIFY_FIRING_TIME(ON-LIST(X))
NEXT
END
```

Table 1. Structural information of the CNC machine model.

No. of eye-glasses to manufacture	SS structural information	No. transitions	No. place nodes
2 Eye-glasses: Type A	No. Nodes: 19,232 No. Arcs: 59,610 No. OLD Nodes: 15,431 Levels: 53	16 transitions	5 place nodes
3 Eye-glasses: Type B	No. Nodes: 172,242 No. Arcs: 765,177 No. OLD Nodes: 145,911 Levels: 84	16 transitions	5 place nodes
3 Eye-glasses: 2 Type A + 1 type B	No. Nodes: 562,799 No. Arcs: 1800,951 No. OLD Nodes: 471,939 Levels: 81	16 transitions	5 place nodes
4 Eye-glasses: 2 Type A + 1 Type B + 1 Type C	No. Nodes: No. Arcs: >2541,330 No. OLD Nodes: >676,223 Levels: 105	16 transitions	5 place nodes

Table 2. Obtained results from the benchmark.

Final marking of the buckets place node	Makespan A^*: $g(x) + h(x)$ $h = 0$	Makespan A^*: $g(x) + h(x)$ $h = -\text{depth}$	Makespan A^*: $g(x) + h(x)$ $h = \text{Min}$ (remaining op. time)	Makespan DFS approach	Makespan TLS VerA-algorithm	Makespan New TLS algorithm
Buckets: 2′(1,1,1,2,6,6,215,220)	853 s	853 s	853 s	1039 s	635 s	590 s
Buckets: 3′(1,1,1,2,6,6,215,220)	1360 s	1360 s	1360 s	1335 s	960 s	833 s
Buckets: 2′(1,1,1,2,6,6,215,220) +1′(2,1,3,4,6,6,120,120)	1018 s	1018 s	1023 s	955 s	821 s	720 s
Buckets: 2′(1,1,1,2,6,6,215,220) +1′(2,1,3,4,6,6,120,120) + 1′(3,1,5,6,6,6,540,540)	1917 s 500 k nodes explored	1917 s 500 k nodes explored	1903 s 500 k nodes explored	1913 s 500 k nodes explored	Unable to reach the objective state	Unable to reach the objective state

Table 3. The results for the job shop 6×6.

	Different nodes: 117,650 Arcs: 605,061 Levels: 37					
Structural information of the $J - S\ 6 \times 6$	A^*: $g(x) + h(x)$ $h = 0$	A^*: $g(x) + h(x)$ $h = -\text{depth}$	A^*: $g(x) + h(x)$ $h = \text{Min}$ (remaining op. time)	DFS algorithm	TLS VerA algorithm	New TLS
MAKESPAN (1st phase only)	82 s	82 s	103 s	161 s	65 s	60 s
Difference with optimal (55 s)	27 s	27 s	48 s	106 s	10 s	5 s

The variable POS records the current position of the pointer of the ON-list, therefore the procedure VERIFY_FIRING_TIME will evaluate only the S-old nodes ahead of the current global clock.

With the new implementations, the initial feasible path will have better time values than the ones that can be obtained using the older algorithm. This characteristic is very important for developing decision support tools that aim at giving fast response with solutions close to the optimal ones.

5.4 *Experimental results*

In order to verify the improvements of the new TLS algorithm, two models have been tested using different algorithms. In the first three algorithms, the DFS is performed using the implementations of the very well-known A^* discussed in (Lee and DiCesare 1994) for the scheduling of FMS; the other algorithms are previous implementations developed by the authors, namely the DFS without informed search and the original TLS algorithm (Mujica and Piera 2010). The benchmark models used to test the performance of the recent developments are an eye-glass CNC machine which must bevel and adjust batches of eye-glasses using three machines and the 6×6 job shop.

In the case of the CNC machine, each eye-glass is composed by two glasses and each one must undergo operations in the three machines; the complexity of operations varies depending on the workload. Since it is a batch process the SS of the possible operations is bounded, for more details of the model see Mujica and Piera (2009). Table 1 presents the key structural information of the CNC machine.

The obtained results for the different workloads using the different algorithms are presented in Table 2.

There were some cases where it was not possible to obtain a result due to the memory saturation thus revealing other limitations which must be faced in order to further improve the algorithm.

The algorithm was also evaluated with the 6×6 job shop benchmark (Table 3) whose optimal value has been reported in literature as 55 time units (Fang, Ross, and Corne 1993). The time values of the objective node obtained at the end of the first phase (all the jobs completed) were compared against the reported optimal values of this benchmark. This comparison enables to realise the efficiency of the algorithm in order to get optimal values in the first phase thus giving insight of its potential for being used for decision support tools in industrial environments.

In both models, the new TLS algorithm clearly outperforms the results obtained by a traditional implementation such as the A^* and also the previous implementations developed by the authors.

6. Conclusions and future work

The scheduling of manufacturing systems is a challenging task due to the combinatorial nature of these ones. An improved algorithm to face with the scheduling problems based on the firing time of timed coloured Petri net transitions has been presented. The algorithm clearly shows an improvement in the capacity to obtain quasi-optimal results in short time which is a key characteristic in any decision support tool for manufacturing management. The implementations aimed at obtaining the minimal makespan which is an important objective in manufacturing and also in other industrial systems. The implementations presented in this article together with the compacted timed SS allow increasing the analysis capabilities of optimisation algorithms based on explorations of the SS of timed coloured Petri nets models. Furthermore, the algorithms presented in pseudo code can be easily implemented in any programming language and the search rules can be extrapolated for managing limited resources in other industrial areas where the total time of a process is the variable to be minimised (e.g. transport systems, aeronautical processes, etc.)

The algorithm has a limitation; that is the memory saturation when the generated information for the SS reaches the computer capacity. In such a case, it will be necessary to perform external storing or developing a garbage collection algorithm for the CTSS; the former can be easily implemented but the performance of the overall algorithm will depend directly on the rate of the I/O data transfer operations to the hard disc; the latter will need further research for the implementation using the CTSS because a garbage collection algorithm for complete descriptions of the markings has been reported (Christensen, Kristensen, and Mailund 2001), however to the best of the knowledge of the authors, there is no implementation for a compacted version of the timed SS.

Another approach for avoiding as much as possible the memory saturation could be the development of an heuristic to control the expansion of the SS such as the one implemented by Mejia and Odrey (2005) for ordinary Petri nets. These implementations are being part of the current research of the authors.

Acknowledgements

This research was supported by the Mexican Council of Science and Technology (CONACYT).

References

Berthomieu, B., P. O. Ribet, and F. Vernadat. 2004. "The Tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets." *International Journal of Production Research* 42 (14): 2741–2756.

Chiola, G., C. Dutheillet, G. Franceschinis, and S. Haddad. 1997. "A Symbolic Reachability Graph for Coloured Petri Nets." *Journal of Theoretical Computer Science* 176 (1–2): 39–65.

Christensen, S., K. Jensen, T. Mailund, and L. M. Kristensen. 2001. "State Space Methods for Timed Coloured Petri Nets." In *Proceedings of 2nd International Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, 33–42, Berlin.

Christensen, S., L. M. Kristensen, and T. Mailund. 2001. "A Sweep-line Method for State Space Exploration." *Lecture Notes in Computer Science* 2031: 450–464.

Cormen, T. H., C. E. Leiserson, and R. L. Rivest. 1990. *Introduction to Algorithms*. Cambridge, MA: MIT Press and McGraw-Hill.

Dauzére-Peres, S., and J. B. Lasserre. 1994. *An Integrated Approach in Production Planning and Scheduling. Lecture Notes and Mathematical Systems*. Berlin: Springer.

Fang, H., P. Ross, and D. Corne. 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*, Urbana-Champaign, IL.

Ghoul, R. H., A. Benjelloul, S. Kechida, and H. Tebbikh. 2007. "A Scheduling Algorithm Based on Petri Nets and Simulated Annealing." *American Journal of Applied Sciences* 4 (5): 269–273.

Huang, B., Y. Sun, and Y. M. Sun. 2008. "Scheduling of Flexible Manufacturing Systems Based on Petri Nets and Hybrid Heuristic Search." *International Journal of Production Research* 46 (16): 4553–4565.

Jensen, K. 1996. "Condensed State Spaces for Symmetrical Coloured Petri Nets." *Formal Methods in System Design* 9: 7–40.

Jensen, K. 1997. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Vol. 1. Berlin: Springer.

Jensen K., T. Mailund, and L. M. Kristensen. 2001. "State Space Methods for Timed Coloured Petri Nets." In *Proceedings of 2nd International Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, Berlin.

Jensen, K., and L. M. Kristensen. 2009. *Coloured Petri Nets: Modeling and Validation of Concurrent Systems*. Berlin: Springer.

Kristensen, L. M., and T. Mailund. 2002. *A Generalized Sweep-line Method for Safety Properties*, 549–567. *FME*. Berlin: Springer.

Lee, D. Y., and F. DiCesare. 1994. "Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search." *IEEE Transactions on Robotics and Automation* 10 (2): 123–131.

Mejia, G., and C. Montoya. 2008. "A Petri Net Based Algorithm for Minimizing Total Tardiness in Flexible Manufacturing Systems." *Annals of Operations Research* 164 (1): 63–78.

Mejia, G., and C. Montoya C. 2009. "Scheduling Manufacturing Systems with Blocking: A Petri Net Approach." *International Journal of Production Research* 47 (22): 6261–6277.

Mejia, G., and N. Odrey. 2005. "An Approach Using Petri Nets and Improved Heuristic Search for Manufacturing System Scheduling." *Journal of Manufacturing Systems* 24 (2): 79–92.

Merkuryev, Y., G. Merkuryeva, M. A. Piera, and T. Guasch. 2009. *Simulation-based Case Studies in Logistics: Education and Applied Research*. London: Springer.

Moore, K. E., and S. M. Gupta. 1996. "Petri Net Models of Flexible and Automated Manufacturing Systems: A Survey." *International Journal of Production Research* 34 (11): 3001–3035.

Mujica, M. A., and M. A. Piera. 2009. "Performance Optimization of a CNC Machine Through Exploration of Timed State Space." In *Proceedings of the International Modelling Multiconference (I3M)*, 20–25, Tenerife, September 23–25.

Mujica, M. A., and M. A. Piera. 2010. "Time Line Search for the State Space-based Optimization Algorithm for Timed Coloured Petri Nets." Proceedings of the IFAC MCPL 2010, Coimbra, September 8–10.

Mujica, M. A., and M. A. Piera. 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.

Mujica, M. A., M. A. Piera, and M. Narciso. 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.

Pinedo, M. L. 1995. *Scheduling: Theory Algorithms and Systems*. New York: Prentice-Hall.

Pinedo, M. L. 2005. *Planning and Scheduling in Manufacturing and Services*. New York: Springer.

Pitts, R. A., and J. A. Ventura. 2009. "Scheduling Flexible Manufacturing Cells Using Tabu Search." *International Journal of Production Research* 47 (24): 6907–6928.

Ravindra, G., and M. Mathirajan. 2011. "Heuristic Algorithms for Scheduling of a Batch Processor in Automobile Gear Manufacturing." *International Journal of Production Research* 49 (10): 2705–2728.

Valmari, A. 1998. "The State Explosion Problem." *Lecture Notes in Computer Science* 1491: 429–528. London: Springer.

Westergaard, M., L. M. Kristensen, G. S. Brodal, and L. Arge. 2007. "The ComBack Method – Extending Hash Compaction with Backtracking." *Lecture Notes in Computer Science Proceedings of Application and Theory of Petri Nets* 4546: 445–464. Berlin: Springer.

Wolf, K. 2007. “Generating Petri Net State Spaces.” In *Proceedings of the 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, 29–42, Berlin: Springer.

Xiong, H. H., and M. C. Zhou. 1998. “Scheduling of Semi-conductor Test Facility via Petri Nets and Hybrid Heuristic Search.” *IEEE Transactions on Semiconductor Manufacturing* 11 (3): 384–393.

Zuniga, C., M. A. Piera, and M. Narciso. 2011. “Revisiting the Pallet Loading Problem Using a Discrete Event System Approach to Minimise Logistic Costs.” *International Journal of Production Research* 49 (8): 2243–2264.