HEURISTIC ALGORITHMS TO OBTAIN OPTIMIZED SCHEDULES OF MANUFACTURE TCPN MODELS

Miguel A. Mujica[1], Miquel Angel Piera[2]

[1,2]Autonomous University of Barcelona, Faculty of Telecommunications and Systems Engineering,
08193 Bellaterra, Barcelona

Miguelantonio.mujica@uab.es (Miguel Mujica)

Miquelangel.piera@uab.es (Miquel Piera)

Abstract

The timed coloured Petri Net formalism is a modelling formalism which allows several abstraction levels depending on the objective of the study. In order to coordinate the main activities of manufacturing systems it is desired to have simulation-based computer tools that help the decision maker to cope with the complex process of schedule all the available resources in the system. The exploration and analysis of the state space of timed Coloured Petri Nets has been used by several authors to evaluate systems behaviour as well as a search space for states of particular interest. In this paper, recent developments of heuristics implemented in algorithms that use the state space to transform the simulated-based optimization problem into a search problem are presented. These algorithms can be implemented in software tools for controlling and coordinate the activities of real-time systems.

Keywords: Coloured Petri Nets, State space, Heuristics, Makespan, Optimization.

Presenting Author's biography

Miguel A. Mújica Mota was born in Mexico City. He studied chemical engineering at Autonomous Metropolitan University of Mexico, an MSc in Operations Research in the National Autonomous University of México. He also received a Master's in Industrial Informatics from the Autonomous University of Barcelona. Actually he is a PhD Student at Autonomous University of Barcelona. He also has professional experience in manufacture and production planning in the cosmetic industry. His research interest focuses on optimization techniques using the Coloured Petri Nets formalism aiming to solve industrial problems.

1 Introduction

In manufacturing processes there exists always the problem of coordinating all the available resources in order to obtain a manufacturing goal (the production of goods). In every production day it results very difficult to adequately coordinate all the different resources to produce the different goods, normally this task is achieved assigning the resources based on a short time horizon taking into account the information at hand (amount of products needed, raw materials and human resources available, etc) instead of evaluating the outcomes of all the possible resource allocation.

Simulation has been traditionally used as a tool which optimizes the available resources configurations through the execution of several simulation experiments which allow evaluating a high quantity of resource allocations. Unfortunately this approach results insufficient to generate an optimal allocation or resource configuration; it produces in the most optimistic case a configuration close to the optimal, but the optimal assignment can be never assured. The Timed Coloured Petri Net Formalism (TCPN) is a modelling formalism which in conjunction with the analysis of the so-called reachability graph can overcome this problem transforming the simulation-based optimization problem into a search problem. The main drawback of using this approach is the *state explosion* [12] which in most of the cases saturates the computer resources without obtaining the optimal configuration or the objective state. This situation makes clear the necessity of developing intelligent exploration approaches which allow exploring the highest possible quantity of nodes without reaching the computer limitations. In this article some algorithms combined with heuristic selection rules are presented. These algorithms have given good results when they were applied in manufacture TCPN models under particular scenarios which will be discussed in the correspondent sections.

1.1 The TCPN formalism

Coloured Petri Nets (CPN) is a simple yet powerful modelling formalism, which allows the modelling of complex systems which present an asynchronous, parallel or concurrent behaviour and can be considered discrete event dynamic systems [4,7]. The formalism allows modelling not only the dynamic behaviour of systems but also the information flow, which is an important characteristic and very useful in industrial systems modelling.

In order to investigate the KPI's (Key Performance Indicators) at which the industrial systems operate under different policies, such as scheduling, resource occupancy, costs and inventory among others it is convenient to extend CPN with a time concept (TCPN). This extension is made introducing a global clock for the model, time stamps for the entities and a time delay for the model transitions. The global clock represents the model time, and the time stamps describe the earliest model time at which the entities of the model, graphically represented by dots (tokens), can be used for the transition evaluation process [6]. A token is *ready* if the correspondent time stamp is less than or equal to the current model time. If the token is not ready, it can not be used in the transition enabling procedure.

It is natural to associate activities of the real system to transitions in the TCPN model. Simulation community use TCPN to specify discrete event systems by attaching a Δr time delay to transitions in order to simulate the time consumption of a certain activity. Therefore when a transition occurs, the output tokens will have a time stamp Δr time units larger than the current global clock *Gc* which simulates time delay due to the execution of an activity. Therefore the time values of the output tokens can be obtained using the following formula:

$$t_o = Gc + \Delta r \quad (1)$$

Where t_o is the time stamp value of the output tokens when the transition firing takes place.

Gc is the global clock of the model when the firing occurs.

Δr is the time associated with the transition.

It is a common convention to use the sign @ to denote time in the elements of the model. When it is attached to transitions, it specifies the time consumption Δr.

The TCPN can be formally defined as follows.

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

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 $P \cap T = \varnothing$

3. $A \subseteq P \times T \cup T \times P$ is a set of directed arcs

4. $\sum$ is a finite set of non-empty colour sets.

5. V is a finite set of typed variables such that Type $[\upsilon] \in \sum$ for all variables $\upsilon \in V$.

6. C: $P \rightarrow \sum$ 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 assigns a time delay to each transition.

10. I is an initialization function assigning an initial timed marking to each place p such that:

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

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

The *state* of every TCPN model is also called the *timed marking* which is composed by the expressions with their time stamps associated to each place *p*.

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* and the expressions contain also the time information (time stamps and global clock).

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

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

1.2 The Compact Timed State Space

The reachability graph or state space (SS) is a directed graph used commonly for the verification and analysis of behavioural properties of CPN models such as liveness, boundedness and reachability among others [5] which determine the behaviour of the model. The analysis is performed through the generation and storage of all the different reachable states from an initial one.

The main characteristics of the timed SS are:

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

The following definitions are necessary to describe the algorithms performance.

Definition 5. Let $\mathfrak{M}^T$ be the set of timed markings of a state space, and M_i^T, M_k^T be timed markings. A state M_k^T will be considered as *old node* if it is exactly the same (together with its time values) as one M_i^T that had been previously generated in any other level of the SS, i.e. $M_k^T = M_i^T$

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

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

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

Some authors have developed different ways of representing the timed sate space (TSS) basing their representations on different characteristics of the model itself [5]; these representations were developed with the purpose of reducing as much as possible the state explosion problem without loosing analysis capabilities. The SS used in this work is the so called *Compact Timed State Space* [8], and it was developed aiming to optimize a utility factor.

The compact timed state space (CTSS) is a particular version of a TSS [8] which reduces the TSS taking advantage of the repeated untimed markings. The latter is carried out during the construction of the SS in the following way. When a timed marking whose underlying untimed marking appears exactly the same as the one of a state previously generated, the marking with the repeated characteristics is not explored again; instead it is marked as *S-old node* and its time elements (time stamps and global clock) stored for later analysis. Let us put this in a more formal way.

Definition 9. Let $\mathfrak{M}^T$ be the set of timed markings of a state space. Let M_i^T and M_k^T be timed markings with their correspondent untimed markings M_i^U and M_k^U.

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

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

The use of the S-old nodes allows generating an SS that can be used to verify system properties and to explore the SS without loosing important time characteristics that invalidate the results obtained.

During the generation of the CTSS when it is detected an S-old node, both states are merged and the discovered one is not explored again thus reducing the state space size and the number of operations to generate it. Figure 1 exemplifies the information contained in the CTSS where node 1 represents the initial marking at 0 time of the TCPN model; the nodes that have several input arcs represent the S-old nodes.

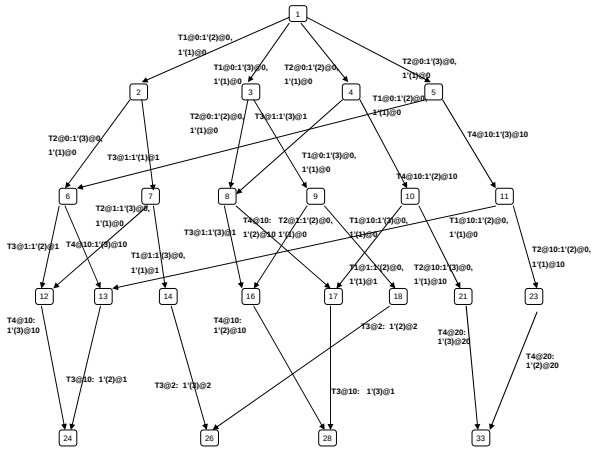
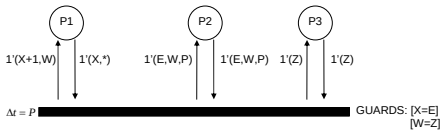


Figure 1. The CTSS of a TCPN model

1.3 The Job-Shop and Open-Shop Scheduling Problems.

The job-shop and open-shop scheduling problems in its different modalities are part of the most difficult NP-hard problems [3]. The job shop scheduling problem (JSSP) consists of *n* jobs and each job visits a number of machines following a predetermined route. In some models a job may visit any given machine at most once and in other models a job may visit each machine more than once. In the latter case it is said that the job shop is subject to recirculation. Job shops are prevalent in industries where each customer order is unique and has its own parameters. There are several industries which perform as a work shop such as wafer fabrics or hospitals [10]. Therefore the solutions to these kind of problems would help to efficiently coordinate the different resources and activities in these kinds of industries. The case of the Open shop scheduling problem (OSSP) involves a collection of *m* machines and a collection of *n* jobs where there is no particular predefined order of the operations, but the machines can be used only once by each job. The main objective in this study is to generate a schedule with a makespan as short as possible; the makespan is simply the total elapsed time in the schedule. To achieve this objective TCPN models whose makespan were optimized using the CTSS together with search algorithms have been developed. Figure 2 exemplifies the kind of models developed in this study.



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

Figure 2. TCPN model of a JSSP

2 Heuristic Algorithms

In this section three different algorithms which gave good results with different job-shops and open shops are presented. In the following subsections the algorithms are briefly presented and their characteristics discussed.

2.1 The two-step algorithm

This algorithm has the characteristics that generate the state space in one phase and the second phase is used for analyzing the timed elements of the S-old nodes found during the generation phase. Some of these algorithms have been discussed in previous works [8] and implemented to solve the 3x3, 5x5 and 6x6 JSSP [2]. The first phase of the algorithm generates the state space using a *depth first search* (DFS) algorithm which can be improved using different heuristics for selecting the successor nodes to be evaluated each time the algorithm evaluates the next level down the tree. It is implemented a utility function f_k which assigns a value to each successor node based on the values of the time stamps and it is selected the successor with the lowest value. Figure 3 exemplifies this principle.

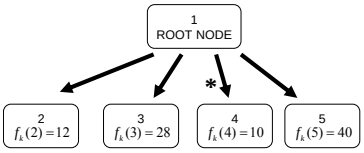


Figure 3. The use of a utility function in the DFS algorithm

During the state space generation, every repeated S-old node is not evaluated again, instead its time elements are stored for being analyzed in the next step. Figure 4 illustrates the generated list when several S-old nodes are detected.

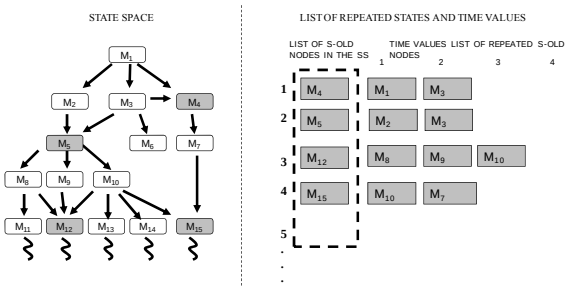


Figure 4. State space and the S-old node list

The left hand side of the figure represents the generated state space, and the right hand side of the figure represents that the original S-old nodes are stored only once, and when it is detected a node that produce a similar marking (but with different time values) the time elements are stored in a list which will be used by the optimizing step of the algorithm.

The second phase of the algorithm evaluates the stored *S-old nodes* generated during the first phase to detect those that optimize the sub-branches that lead to the optimal makespan of the original TCPN model.

The S-old node analysis is carried out performing the following procedure.

Let $M_i^T, M_j^T \in \mathfrak{M}^T$ and the corresponding list of time stamps of each node T_i and T_J. Function Φ_k is used to make the time comparison between time stamps in the following way.

Let Ψ be an auxiliary function such as

$$\begin{aligned} \Psi: \ \mathbb{R}\times\mathbb{R} &\rightarrow \{0,1\} \\ (x,y) &\mapsto \begin{cases} 0, x \geq y \\ 1, x < y \end{cases} \end{aligned} \tag{2}$$

with the use of the auxiliary function, the comparison is carried out applying formula (3).

$$\begin{aligned} \Phi_k: \ \mathbb{N}^k\times\mathbb{N}^k &\rightarrow \mathbb{N} \\ (T_i, T_j) &\mapsto \sum_{l=1}^{k} \Psi(T_{il}, T_{jl}) \end{aligned} \tag{3}$$

Applying the function Φ_k to the S-old nodes M_i^T and M_j^T where M_i^T is the S-old node that appeared originally in the TSS and M_j^T the S-old node with different time stamps, the following three possible outcomes can be obtained:

Φ_k = 0: The time stamp values of the M_i^T state are equal or higher than the ones of the M_j^T state. In this case the best branch is chosen from among the successors of M_i^T (since the best branch of M_i^T will also be the best one for M_j^T) 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^T state. The previous procedure is done using token and transition information stored from the generation of M_i^T.

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

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

2.2 Time-Line algorithm

This algorithm was developed based on the idea of the so called *sweep line method* [1].This algorithm performs also in two phases. In the first phase the CTSS is generated using a firing time-basis. The nodes to be evaluated are selected based on the time value when the first firing occurs, i.e. the earliest firing time.

In this case the nodes to be explored are selected based on the following procedure.

Let $M_i^T \in \mathfrak{M}^T$ be a timed marking, T_i its correspondent time stamp list and Gc the global clock of the timed marking. The function $tline_k$ assigns a value or key to each fired state

$$\begin{aligned} tline_k: \ \mathbb{N}^k\times\mathbb{N} &\rightarrow \mathbb{N} \\ (T_i, Gc) &\mapsto Max\{Min\{T_{i1},...,T_{ik}\}, Gc\} \end{aligned} \tag{4}$$

Using formula (4) the nodes are being evaluated when the value matches a variable that has the current firing time; so the nodes with the same firing time will be evaluated first independently of the branch they belong to. Under this perspective several branches are explored at the same firing time and the objective node is found at the end of the generation phase with a makespan close to the optimal one. The drawback of

this approach compared to the two-step algorithm is that the two-step one uses a DFS algorithm so it can get results (not necessarily the optimal ones) when it faces big state spaces without exploring the whole state space. On the contrary, the time-line algorithm requires to store in computer memory all the information related to the nodes of the different branches explored so far to maintain the time coherence of the different branches of the state space.

The second phase of the time line algorithm performs exactly the same way as the two-step algorithm evaluating the S-old nodes generated so far.

2.3 Hybrid approach

A hybrid approach which has the benefits of both previous approaches (cope with huge state spaces and evaluation of several branches at once) was also developed. With the use of this approach the first phase is decomposed in two sub phases where the first one uses a DFS algorithm to find a feasible path. In this case heuristic rules that lead the search can also be implemented. When the objective state has been found, the generation algorithm switches to a time-line algorithm so several branches can be evaluated at the same time. The importance of having an initial feasible path is because the optimization step requires an objective node in order to optimize it. The hybrid algorithm will be composed of three main phases:

Phase A: The algorithm explores the CTSS under DFS logic until it finds the objective node. When the objective state is found, it stores the feasible path.

Phase B: The algorithm switches to a TLS logic in order to generate S-old nodes which result useful for optimization purposes. This phase continues until CPU resources are close to their limits.

Phase C: The generated S-old nodes are analyzed for optimizing the objective state.

It is important to mention that since the time line algorithm selects the nodes based on the earliest firing time, it is expected that the solution is very sensitive to the initial assignments; the latter has been reported by some authors[9,10]. Based on that assumption, it is expected that the solution found by the hybrid approach is performed by analyzing the S-old nodes that are in the lower levels of the reachability tree, this characteristic results very useful when it is not possible to keep in memory the complete state space of the studied problem.

3 Experimental Results

The presented approaches have been implemented in three job-shops and in one open-shop. The results of the implementation in the three job-shops are presented in Table 1 which resumes the performance of the different algorithms with these kind of problems.

Table 1. Algorithm Performance Indicators

Type of Model	Algorithm Type	Obtained Make Span	Stored S-Old Nodes	Updated Nodes after Stamp Evaluation	Rejected Nodes After Stamp-Evaluation	Undecided Nodes after Stamp Evaluation	Updatings Performed after depth exploration	Type of Analysis
JS 3x3	Two-Step Algorithm	15 time units	2,712	100	1,255	1,356	4	Complete
JS 3x3	Time-Line Search	15 time units	2,712	5	1,240	1,467	0	Complete
JS 3x3	Hybrid Algorithm	15 time units	2,712	18	1,259	1,435	0	Complete
JS 5x5	Two-Step Algorithm	428 time units	32,375	4,642	15,160	12,385	41	Complete
JS 5x5	Time-Line Search	428 time units	32,375	708	16,394	15,272	6	Complete
JS 5x5	Hybrid Algorithm	428 time units	32,375	685	16,286	15,403	25	Complete
JS 6x6	Two-Step Algorithm	255 time units	605,020	64,466	303,111	237,173	41	Complete
JS 6x6	Time-Line Search	255 time units	605,020	14,242	277,128	313,559	5	Complete
JS 6x6	Hybrid Algorithm	255 time units	605,020	8, 324	293,423	303,273	19	Complete
OS 4x4	Two-Step Algorithm	191 time units	1,805,613	185,413	867,214	752,986	92	Partial (600,000 nodes)
OS 4x4	Time-Line Search	----	----	----	----	----	----	Unable to Analize the Problem
OS 4x4	Hybrid ALgorithm	196 time units	1,805,613	20,213	1,204,617	580,783	12	Partial: (600,000 nodes)

It can be noticed that in the case of the job shop problems the three algorithms give the same optimized make span, but the key performance indicators vary from every type of model. It has been reported by the authors[8] that the most time consuming activity is the analysis of the results that come from the second outcome of the analysis (section 2.1). From the results it is clear that the time line algorithm is the one that gives feasible solutions closest to the optimal because when the analysis is performed, the quantity of updatings (column 8 in the figure) is small compared to the ones performed by the two-step algorithm. In the case of the hybrid algorithm the performance falls in the middle of both algorithms.

It is fair to mention that the obtained results for the job shops correspond to the ones reported in literature [2] which confirm the value of the presented algorithms when these type of problems are faced. Due to the size of the job shop problems it was possible to generate the complete CTSS and evaluate the performance of the three algorithms. In the case of a very hard problem such as the open shop 4x4 [11] it was impossible to maintain in computer memory the total amount of states. With problems like the open shop the two-step algorithm and the hybrid approach are capable of getting to the objective node without saturating computer memory but with the limitation that they can only perform a partial exploration.

4 Conclusions and Future Work

Three algorithms that combine simulation with search algorithms implemented in state space analysis have been presented. Depending on the rules to explore the state space these algorithms are capable of facing different size problems. Those that follow a DFS analysis are able to cope with problems whose state

space is so big that result unfeasible to maintain in computer memory all the state space information but they are able to produce feasible solutions that are expected to be close to the optimal ones. On the other hand the algorithms that evaluate several branches at a time produce solutions closer to the optimal ones but they have the disadvantage that they require to maintain in computer memory all the generated information to perform the analysis of the S-old nodes which optimize the feasible path. The performance indicators suggest also that the performance of the TLS algorithm requires less analysis of the stored S-old nodes therefore it is possible to improve even more the performance of these algorithms in order to develop decision support tools that give response in little time.

The authors are actually working on an algorithm that not only maintains the advantages of the combination of both approaches but also that throws away the information that is not useful for optimization purposes so that it avoids the memory saturation in most computers.

5 References

[1] Christensen S., T. Mailund, 2002, A Generalized Sweep-Line Method for Safety Properties, in *FME*, Springer- Verlag, Berlin- Heidelberg

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

[3] Gary M.R., Johnson D.S., 1979, Computers and Intractability: A Guide to NP-Completeness, Freeman

[4] Jensen K., 1997, Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Vol.1 Springer-Verlag. Berlin.

[5] Jensen K., T. Mailund, L.M. Kristensen,2001, State Space Methods for Timed Coloured Petri Nets, in Proceedings of 2nd Intenational Colloquium on Petri Net Technologies for Modelling Communication Based Systems, Berlin

[6] Jensen K., Kristensen L.M., 2009, Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer.

[7] Moore K.E., S.M. Gupta, 1996, Petri Net Models of Flexible and Automated Manufacturing Systems: A Survey, *in International Journal of Production Research,* Vol.34, No. 11

[8] Mujica M.A., Piera M.A., 2010, Revisiting state space exploration of timed coloured petri net models to optimize manufacturing system's performance, *in Simulation Modelling Practice and Theory*, Vol.18, 9, p.p. 1225-1241.

[9] Music G., 2009, Petri Net Based Scheduling Approach Combining Dispatching Rules and Local Search, in *Proceedings of the I3M2009 Multiconference*, Tenerife, Spain

[10] Pinedo M.L., 2005, Planning and Scheduling in Manufacturing and Services, Springer, New York

[11] Taillard, E., 1993. Benchmarks for basic scheduling problems, *European Journal of Operational Research*

[12] Valmari A., 1996, The State Explosion Problem, in *Lecture Notes in Computer Science*, Vol.1491, Springer-Verlag, London