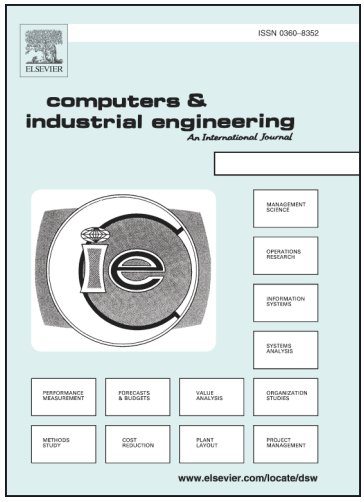
Accepted Manuscript



TIMSPAT – Reachability graph search-based optimization tool for colored Petri net-based scheduling

Olatunde T. Baruwa, Miquel A. Piera, Antoni Guasch

PII: S0360-8352(16)30266-2
DOI: http://dx.doi.org/10.1016/j.cie.2016.07.031
Reference: CAIE 4428

To appear in: *Computers & Industrial Engineering*

Please cite this article as: Baruwa, O.T., Piera, M.A., Guasch, A., TIMSPAT – Reachability graph search-based optimization tool for colored Petri net-based scheduling, *Computers & Industrial Engineering* (2016), doi: http://dx.doi.org/10.1016/j.cie.2016.07.031

TIMSPAT – Reachability graph search-based optimization tool for colored Petri net-based scheduling

Olatunde T. Baruwa[a,*], Miquel A. Piera[a], Antoni Guasch[b]

[a] *Logistics and Aeronautics Unit, Department of Telecommunications and Systems Engineering, Universitat Autònoma de Barcelona, 08202, Campus Sabadell, Barcelona, Spain*

[b] *Departament d'Enginyeria de Sistemes i Automàtica, FIB (Campus Nord), Universitat Politècnica de Catalunya, 08034 Barcelona, Spain*

Abstract

The combination of Petri net (PN) modeling with AI-based heuristic search (HS) algorithms (PNHS) has been successfully applied as an integrated approach to deal with scheduling problems that can be transformed into a search problem in the reachability graph. While several efficient HS algorithms have been proposed albeit using timed PN, the practical application of these algorithms requires an appropriate tool to facilitate its development and analysis. However, there is a lack of tool support for the optimization of timed colored PN (TCPN) models based on the PNHS approach for schedule generation. Because of its complex data structure, TCPN-based scheduling has often been limited to simulation-based performance analysis only. Also, it is quite difficult to evaluate the strength and tractability of algorithms for different scheduling scenarios due to the different computing platforms, programming languages and data structures employed. In this light, this paper presents a new tool called TIMSPAT developed to overcome the shortcomings of existing tools. Some features that distinguish this tool are the collection of different HS algorithms, the event-driven exploration of the timed state space including its condensed variant, localized enabling of transitions, the introduction of a static place, and its easy-to-use syntax statements. The tool is easily extensible and can be integrated as a component into existing PN simulators and software environments. A comparative study is performed on a real-world eyeglass production system to demonstrate the usage of the tool for scheduling purposes.

Keywords: Colored Petri net, Scheduling, Reachability graph, Heuristic search, Timed state space, Flexible manufacturing system, Simulation

*Corresponding author

Email addresses: olatundetemitope.baruwa@e-campus.uab.cat, tbaruwa@gmail.com (Olatunde T. Baruwa), miquelangel.piera@uab.es (Miquel A. Piera), toni.guasch@upc.edu (Antoni Guasch)

1. Introduction

In the manufacturing industry, companies are consistently faced with the challenge on how to maximize the utilization of limited resources to perform a collection of tasks (jobs) while optimizing a certain performance measure to meet customer due dates given the changing customers demands and tight production requirements. Efficient planning and scheduling policies are critical to the survival of companies in today's globally competitive market. Due to the complexity and level of detail required by real-world systems, optimization of scheduling problems are known to be NP-hard since the computation time to obtain an optimal schedule grows exponentially with the problem size [70]. While most solution approaches are largely dominated by optimization models based on mathematical programming, there has been a consistent rise in the use of formal modeling techniques such as timed Petri nets (TPNs) [7, 63] and timed Automata (TA) [27, 54] for planning and scheduling. Besides their capability to validate and verify the behavior of systems, the simulation capabilities of these formal methods make it more flexible in combining them with solution approaches from Operations Research, Artificial Intelligence (AI), and the Computer Science domains. Specifically, Petri nets (PNs) are a powerful graphical and mathematical modeling tool, which have been extensively used to model, simulate, and analyze discrete-event systems characterized by concurrency, parallelism, causal dependency, resource sharing and synchronization [10, 52]. Since its inception in Carl Adam Petri's PhD dissertation on "Communication with Automata" in 1962, it has gained recognition in the research community in addressing manufacturing and logistic systems including its application in a number of different disciplines like communications, transportation, robotics, engineering, business and air traffic management [80].

This paper presents a TIMed State space Performance Analysis Tool (TIMSPAT) for the modeling and analysis of scheduling problems described by the timed colored PN (TCPN) formalism based on the integrated PN and heuristic search (PNHS) scheduling methodology. TIMSPAT is a tool developed as part of the PhD thesis in [4]. To the best of our knowledge, this is a first attempt at providing a scheduling tool for the optimization of TCPN models with reachability graph search-based HS methods. Thanks to the common data structure of AI HS methods, the tool is capable of incorporating several search algorithms in a single executable. So far, nine algorithms have been implemented, ranging from the classical A* [60], hybrid heuristic search [31, 49, 50, 59] to anytime algorithms [6, 9, 47, 74, 75].

In the PNHS approach, the main idea is to formulate the optimization of a scheduling problem as an AI automated planning problem [14, 26] that takes as input the T(C)PN model, the initial state, and a desired goal state, and then produces a sequence of actions to achieve the goal (complete schedule) using heuristic search. Here, the scheduling problem is transformed into a search-based optimization problem in the reachability graph (or state space) of finding the optimal or near-optimal sequence of transition firings from some initial state to the goal state, which minimizes some performance measure. Reachability graph (RG) analysis is a reliable and efficient method to obtain optimal schedules since it can be used as an automated decision support tool to generate all the possible alternatives of the system configuration. A basic intuition underlying the use of RG is that all the reachable states in the T(C)PN are represented as nodes, and the transformation of these states that triggers a change in the system state, as edges. However, the state explosion problem has limited its practical applicability.

AI HS methods [29, 42, 49, 59, 71, 72] have been proposed to simulate only the best scenarios by generating partial RGs with heuristic functions (to guide the search) that rely on the knowledge of the production plans. From Tuncel & Bayhan [63]'s review, the PNHS methodology has proved to be an efficient method for solving production scheduling problems. However, majority of the works on PNHS have practically used TPN only. Although the focus is on manufacturing systems, the methodology can be applied to a broad class of scheduling problems since PN is not driven by a problem domain [20]. Recent studies have demonstrated the capability of the PNHS approach to deal with realistic industrial engineering applications such as the train rescheduling problem for a Dutch railway network [67] and resource-constrained project scheduling in the animation and videogame industry [48].

Most model checking tools [35, 36] can be extended to solve the optimal reachability problem for scheduling, where the goal state is the specification of the property to be verified in the model checking problem [17, 44]. However, these tools have not been enabled with optimization capabilities, because verification algorithms are designed to perform an exhaustive exploration of the RG with untimed nets. Although HSF-SPIN [22] is

targeted at directed model checking with HS methods. On the other hand, PNHS scheduling requires only a partial exploration of the state space of timed nets to find an optimal or near-optimal solution guided by heuristic functions. The use of HS-RG algorithms appears to be a well-developed method (in tools) for other modeling formalisms like TA and Promela. State-of-the art tools like SPIN [61], UPAAL-CORA [12, 13], and TAOpt [55, 62] have been used for scheduling in job shops, process systems engineering and chemical production. However, no attempt has been made for TCPN-based scheduling.

The optimization of stochastic nets is not considered in this paper due to the continuous time domain and memoryless property of exponential time distributions [77] that does not make them amenable to PNHS. An alternative approach that is well suited for stochastic nets is the optimization by means of simulation [40] where tools like TIMENET [15, 77], GreatSPN [3], PIPE2 [21], and CPN Tools [32] can be used. Notwithstanding, deterministic PNs have been proved to be successful for applications in uncertain and dynamic environments [48, 67] both as simulation and optimization models using the reactive scheduling strategy where rescheduling can be used to handle unexpected events or disruptions.

This paper is structured as follows. The remaining parts of Section 1 discusses the motivation and the state-of-the-art review on PN-based tools. Section 2 describes the TIMSPAT architecture and its main components in detail. Section 3 presents the case study of the flexible manufacturing cell and its corresponding CPN model. Section 4 reports the computational and benchmarking results of the HS algorithms implemented on the case study considering several production scenarios, while Section 5 concludes the paper and presents the future plans in place to improve the tool's robustness.

1.1. Motivation

In a dynamic manufacturing environment, production managers are usually confronted with constantly changing scheduling scenarios in their day-to-day activities on the shop floor. Different scenarios may arise as a result of changes in the production mix, product types, due dates, part shortages and unanticipated events like machine failure. The availability of a number of solution algorithms can allow production managers make better decisions considered acceptable for each scenario. However, not all existing algorithms can be suited to all kinds of scheduling problems that arise on the shop floor. While it is possible to adapt an algorithm to different production scheduling scenarios, it may turn out to be inefficient for some. Putting different algorithms at the disposal of the production managers given the situations they are best suited for, may go a long way to aid their decision making. Therefore, it is important to provide the decision makers with a platform that supports a neutral representation in which the different solution algorithms could be automatically tested to select the best solution reached or the best algorithm suitable to solve the given problem at hand. However, there is a lack of decision support tools based on PNHS that can afford the aforementioned concept.

One of the advantages presented by the PNHS approach is that different search algorithms can be implemented to evaluate the best schedule of a particular manufacturing scenario. Several heuristic search methods have been developed for PNHS [29–31, 42, 49, 59, 71, 72]. However, it is quite difficult to evaluate and benchmark the efficiency of these algorithms in terms of time and solution quality due to the different computing platforms, programming languages and data structures used.

In spite of the number of readily available software tools for TCPN simulation, not much has been done in its combination with AI HS methods. Simulation is deemed insufficient to evaluate the different alternatives of a system. For decision making purposes, it is only capable of exploring a limited number of scenarios when applied to improve the system performance. Due to the inherent flexibility, using simulation solely for the optimization of scheduling problems with a large number of decision variables would require a large number of runs [66]. One of the issues frequently encountered with TCPN implementation is the handling of the complex data structure for the computation of enabled bindings, transition firing, and storage [33]. This makes the performance analysis of TCPNs quite difficult using state space analysis. Apart from simulation limitations, existing tools have some drawbacks regarding the execution of TCPNs. The time concept defined in the TCPN standard [33]uses the eagerness-to-fire property for transition firings based on the global clock (model time) [25, 57]. This concept imposes synchronization constraints on event occurrence. As a consequence, it limits the level of concurrency exhibited in asynchronous systems like flexible manufacturing systems

Table 1: PN tools and features

Tool/Feature	TPN-Tools	MARIA	PNetLab	GPenSIM	PN Toolbox	CPN Tools	ASAP	COAST
Untimed net	✓	✓	✓	✓	✓	✓	✓	✓
Timed net	✓	✗	✓	✓	✓	✓	✗	✗
Colored net	✓̸	✓	✓̸	✓̸	✗	✓	✓	✓
Simulation (global clock)	✗	✗	✓	✓	✓	✓	✗	✗
Simulation (event-driven)	✓	✗	✗	✗	✗	✗	✗	✗
Exhaustive search (RG)	✓	✓	✓	✓	✓	✓	✓	✓
Heuristic search (RG)	✗	✗	✗	✗	✗	✗	✗	✗
SS reduction methods	✗	✓	✗	✗	✗	✗	✓	✓
Model checking	✓	✓	✗	✗	✗	✓	✓	✗
Planning and scheduling	✗	✗	✗	✓	✗	✓	✗	✓

✓- fully supported, ✗- not supported, ✓̸- partially supported, SS - state space

(FMSs), and thus, precludes the generation of firing sequences that would lead to an optimal schedule. The occurrence of an event should not be restricted by time constraints.

1.2. Related PN tools

There are quite a handful of graphical and command line PN tools available for modeling and simulating discrete-event systems. The PN tool database [56] lists most of the existing general and special purpose tools. Owed to the large number, we do not intend to provide a comprehensive evaluation. Our aim is to highlight those tools that closely matches the requirements of the PNHS approach. Four search criteria were used for filtering the tool list using the following keywords from the database search tool: *State spaces*, *Petri nets with time*, *high level Petri nets*, *simple performance analysis*. Of the 85 tools registered in the database, only 32 implement state space analysis. Among these 32, 13 support timed nets, 6 of which support TCPN, while 4 implement a kind of performance analysis technique. As the requirements get stronger, the list keeps reducing. Most of the developed tools are graphical editors (66) that implements token game animation (46). Only four tools passed the filter: CPN Tools [32], INA, JFern, Petruchio. However, none of these tools has directly supported the PNHS approach. Some of these tools are no longer actively maintained. In general, they implement RG analysis but mainly for model checking used in validation and verification [14]. Since 2011, the model checking contest for PNs [35] benchmarks the efficiency of several state-of-the-art tools on verification techniques using RG for different classes of models. Among these tools are GreatSPN, SMART, AlPINA, Cunf, ITS-Tools, LoLA, TAPAAL, MARCIE, Neco, SARA. The details of these tools can be found on the model checking contest website [35]. Most of them have also been integrated in software environments like Cosyverif and CPN-AMI. Our focus is on tools enhanced with RG capability as well supporting TCPN. Here, PN editors do not fit in neither do token game simulators.

Table 1 shows examples of tools that support RG analysis and/or simulation, taking into account the type of net supported, search algorithms, analysis, and their applications to planning and scheduling. Nearly all the tools are well developed for simulation in addition to having a GUI editor. TPN-Tools [79] is one of the earliest tools that contains several collection of tools developed for performance analysis of TPNs using RG and structural (invariant) analysis. TPNsim [78] is used for event-driven simulation in case the two previous analysis are not suited to solve some classes of models like stochastic nets. MARIA [46] is a RG analyzer for high level algebraic system nets. Tools like PNetLab [11] and SimQPN [37] have been employed for the control and scheduling of manufacturing systems, and queueing systems respectively. Tools like GPenSIM [18, 19] and PN Toolbox [34] are embedded into third-party commercial software packages like MATLAB. Users are required to learn the MATLAB language to develop models. While PNetLab is standalone, it generates a simulator executable each time a new model needs to be run. The colored nets of some PN simulators (PNetLab, GPenSIM) have lesser expressive power than the CPN standard in CPN Tools [33] for example.

CPN Tools stands out as an industrial strength tool that provides both a graphical editing interface and an interactive simulator for constructing and analyzing models. CPN Tools has been used to analyze some scheduling problems [1, 2, 28]. However, the literature reports its usage for simulation-based performance analysis only. Notwithstanding, CPN Tools has a state space analysis plug-in but it has several limitations to support the timed state space exploration of TCPNs [57]. In addition, the absence of efficient search algorithms has limited its applicability. Only basic traversal algorithm like breadth-first search is implemented in the tool, which cannot scale up to industrial-sized problems. Due to the limitations of the state space plug-in, an extensible platform to CPN Tools called ASAP [68] was developed to provide an implementation support for a wide range of advanced state space methods. Yet, the state space methods are primarily aimed at model checking of untimed CPNs. Even with this extension, it is still difficult to integrate one's search algorithm. Both tools (CPN Tools and ASAP) rely heavily on a proprietary functional programming language, the Standard Markup Language (SML). The steep learning curve of SML makes it hard to use the platform to develop algorithms without having gotten a full grasp of the language.

COAST [38] is a closely-related PNHS tool integrated with CPN Tools for specification and scheduling tasks in a course of action to support human planners. However, it employs untimed CPNs that may not be appropriate for scheduling and implements exhaustive search algorithms that can be time-consuming as well as prone to the state space explosion problem.

To overcome these shortcomings, TIMSPAT has the following distinguished features from the existing ones:

- Dedicated standalone tool written in C++ for PNHS approach based on RG analysis.
- Implements an event-driven state space that overcomes the shortcomings of the global clock synchronization for optimization. The state space can be explored either as an earliest timed state space (ESS) or condensed ESS [6].
- Offers a localized enabling of transitions. Each transition structure is created as an object and only the places required for enabling the transition are specified.
- Easy-to-write syntax expressions without the need to learn a programming language. Complex mathematical expressions are supported in a plain language format. The syntax combines the standard token expression of CPN and the strength of C++ syntax for marking description, mathematical and customized function expressions.
- Implements a new feature called static place to avoid memory clogging and carrying over constant data from one reachable state to the other.
- Collection of heuristic search algorithms (space, time and space-time efficient).
- Offers a portable and standalone modeling syntax specification that can be easily integrated with other applications or used by anyone willing to implement its own search algorithm.
- Allows for interoperability and extensibility with existing PN simulators using an XML model-based integrator.

2. TIMSPAT architecture

The architecture of TIMSPAT shown in Fig. 1 has three major blocks; the XML model integrator, the simulator, and the search module.The TCPN serves as the input to the tool. Its structure is emulated using text files whose definitions conform to the TIMSPAT syntax specification (TIMSPATLib). The solutions to the TCPN are generated by the search algorithms via the state space storage. The main components are:

- Syntax checker: It validates the specification of the definition files to ensure that it is consistent with the TIMSPATLib syntax instructions. The checker reports errors encountered in files and the files are passed to the simulator only if they have been certified okay.

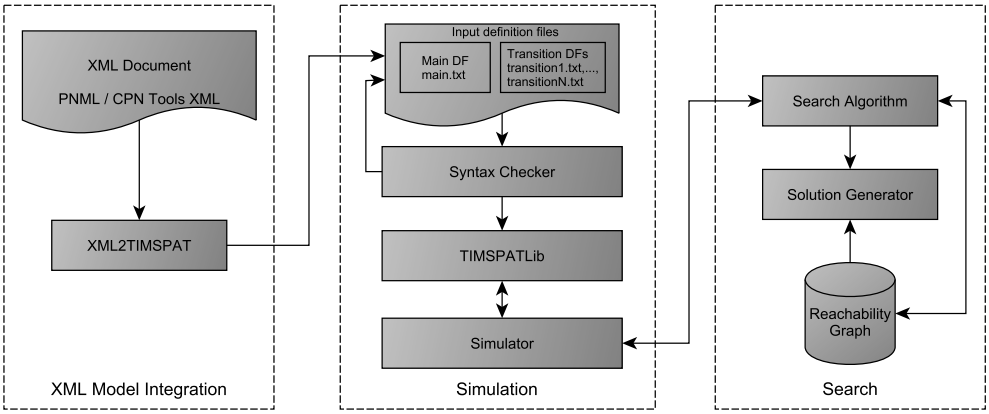


Figure 1: The architecture of TIMSPAT.

- TIMSPATLib: It is a syntax description language implemented as a C++ library for the specification of TCPNs in a textual format. TIMSPATLib interprets and stores the net structure in memory as specified in the definition files.
- Simulator: It performs the discrete-event simulation of TCPNs and can act both as a simulator and a successor generator. The execution is synchronized with the search algorithm module and interfaced with the TIMSPATLib on a continual basis until termination. It uses the stored TIMSPATLib net information to evaluate the states of the system required by the search algorithm.
- Search algorithm: It is used to construct the state space according to a defined objective function. The search algorithm drives the exploration of the state space toward a near-optimal or optimal solution. Solutions are generated from the state space as a sequence of transition firings when a goal marking is reached. They are generated either continuously (when improved solutions are obtained) or at termination depending on the search algorithm employed.
- XML2TIMSPAT: For interoperability, XML2TIMSPAT serves to support the integration of TCPN representation in XML-based file formats from external PN tools for conversion into TIMSPAT syntax format. It parses XML net files and generates their corresponding TIMSPAT definition files. XML2TIMSPAT currently supports the CPN Tools XML and the Petri Net Markup Language (PNML) standard [58]. The model integrator is a standalone executable since post-processing is still required to include TIMSPAT's specific features not currently supported by other tools, such as goal marking representation, objective functions among others.

2.1. TIMSPATLib for TCPN modeling

A CPN is a directed bipartite graph with two node types called places and transitions where the nodes are connected via directed arcs. CPN combines the modeling power of PNs with the strength of programming languages through the definition of data types (color sets) and the manipulation of their data values (colors) [33]. The use of colors allows a concise representation of the system by reducing the graphical complexity of PN models as well as the specification of key characteristics and information flow. A place can contain tokens and is used to describe resources in the system while a token consists of one or more colors describing the entity attributes. Each token can carry a weight called cardinality. A transition describes the event (the start or completion) that may occur (or fire) based on the preconditions of input arc expressions and guards. Graphically, places, transitions, arcs, and guards are represented by circles, boxes, arrows, and square brackets.

Evaluating the performance and investigating different scheduling strategies with ordinary PN or CPN is usually not sufficient. Ordinary PNs/CPNs are extended with time representation where each token

carries a time attribute called time stamp and time delays are associated with transitions, places, or arcs. The time stamp describes the earliest time at which a token becomes available. The introduction of time in CPN (TCPN) makes it possible to capture the temporal behavior in addition to describing the logical behavior and structure of discrete-event systems in a concise manner. With TCPN, quantitative measures like delays, durations and deadlines can be explicitly described. Time can be specified as either deterministic, non-deterministic (interval timed), or stochastic [64]. We assume that the reader is familiar with TCPN formulations and theory [33].

In TIMSPATLib, the TCPN specification for scheduling $TCPN_{SCH}$ is an extension of the TCPN standard [33] with slight modifications on color set definitions and variables in addition to transition delays and goal state.

Definition 1. *Formally, a $TCPN_{SCH}$ for TIMSPAT is defined as $TCPN_{SCH} = (TCPN, M_0, G_M)$ where $TCPN$ is the structure defined as a 9-tuple, $TCPN = (P, T, A, \sum, V, C, G, E, D)$ where:*

1. *P is a finite set of places $\{p_1, p_2, \ldots, p_m\}$.*
2. *T is a finite set of transitions $\{t_1, t_2, \ldots, t_n\}$ such that $P \cap T = \emptyset$.*
3. *A is a finite set of directed arcs $\{a_1, a_2, \ldots, a_k\}$ such that $A \subseteq P \times T \cup T \times P$.*
4. *$\sum$ is a finite set of nonempty types called colored sets that defines the number of token elements (colors) and the operations and functions that can be used in the net inscriptions (i.e. arc and initialization expressions). Each color set is either timed or untimed and an untimed set is either static or otherwise.*
5. *V is a finite set of variables of numeric data types (integer or real).*
6. *$C : P \rightarrow \sum$ is a color set function that assigns a color set to each place. A place p is timed if $C(p)$ is timed; otherwise, p is untimed or untimed static.*
7. *$G : T \rightarrow EXPR_v$ is a guard function that assigns a guard to each transition t such that $Type[G(t)] = true$ or $false$.*
8. *$E : A \rightarrow EXPR_v$ is an arc expression function that assigns an arc expression to each arc a such that $Type[E(a)] = C(p)_{MS}$ if p is untimed or untimed static and $Type[E(a)] = C(p)_{TMS}$ if p is timed, where p is the place connected to the arc and C_{MS} denotes the set of all multisets over C.*
9. *$D : T \rightarrow \mathbb{R}_0^+$ is a time-delay function associated with each transition $t \in T$. It describes the set of firing durations (transition delays). $\mathbb{R}_0^+$ denotes the set of all positive real numbers including zero.*

M_0 is the initial timed marking defined by $M_0(p) = I(p)\langle\rangle \ \forall p \in P$. $I : P \rightarrow EXPR_\emptyset$ is an initialization function that assigns an initialization expression to each place p such that $Type[I(p)] = C(p)_{MS}$ if p is untimed or untimed static and $Type[I(p)] = C(p)_{TMS}$ if p is timed, and
G_M is the set of untimed goal markings (or desired final markings) indicating the completion of a schedule, defined by $M_g(p) = F(p)\langle\rangle \ \forall p \in P$. $F : P \rightarrow EXPR_\emptyset$ is a finalization function that assigns a finalization expression to each place p such that $F(p) = C(p)_{MS}$.

As in CPN Tools, $EXPR$ denotes the set of expressions provided by the TIMSPAT library, and $Type[e]$ denote the type of an expression $e \in EXPR$, i.e., the type of the values obtained when evaluating e. The set of free variables in an expression e is denoted $Var[e]$. A free variable is a variable which is not bound in the local environment of the expression [33].

Note that in this definition, variables do not belong to $\sum$ since variable types (integer and real) are handled by default in the library. As a result, variable definitions are not needed. Only the color set description for each place is required.

The current state of the system is defined by the distribution of tokens over the places called marking. An untimed marking M_u maps each place into a multiset of tokens $M(p) \in C(p)_{MS}$ [33]. TIMSPATLib adopts the functional token expression of CPN Tools.

Definition 2. *A multiset (MS) m over a non-empty set $S = \{s_1, s_2, s_3, \ldots\}$ is a function $m : S \rightarrow \mathbb{R}^+$ that maps each element $s \in S$ into a non-negative integer $m(s) \in \mathbb{R}^+$. It is written as a sum using a single $+$ and $'$:*

$$\sum_{s \in S} m(s)'s = m(s_1)'s_1 + m(s_2)'s_2 + m(s_3)'s_3 + \ldots \tag{1}$$

The non-negative integer $m(s)$ is the number of appearances of the element s in the multiset m. $m(s)$ is also called the cardinality (weight) of the token s.

An element s is a member of a multiset m if the number of appearances $m(s)$ of s in m is greater than zero, i.e., if $m(s) > 0$. The size of a multiset $|m|$ is the sum of the number of appearances of the elements in m, the number of tokens in a place p.

The other operations like addition, scalar multiplication, comparison and subtraction are defined in [33]. The set of all multisets over S is denoted as S_{MS}. The empty multiset over a set S, $\emptyset_{MS}$ is defined as $\emptyset_{MS}(s) = 0, \forall s \in S$.

The multiset of tokens in an untimed place is constructed using the single sum operator + rather than the double ++ in CPN Tools. A timed place attaches a time stamp to each token. For scheduling purposes (see Section 2.3), the time stamps of tokens in a timed place are also constructed as a multiset using the single sum operator + and the @ symbol. Here, the timed multiset of CPN Tools is not used.

The multiset of token time stamps is expressed as:

$$\sum_{s \in S} m(s)@tm[s] = m(s_1)@tm[s_1] + m(s_2)@tm[s_2] + m(s_3)@tm[s_3] + \ldots \tag{2}$$

where $tm[s]$ is the ordered time stamp list of s, $tm[s] = [ts_1, ts_2, \ldots, ts_{tm(ts)}]$ and $ts_i \leq ts_{i++}, \forall i \in \{1, \ldots, tm(s)\}$. It contains the time values $ts \in TS$ for which $m(s) \neq 0$. The @ symbol is omitted if $m(s) = 1$ or $tm[s]$ contains unique values.

As a consequence, the description of tokens in a place has two parts: the set of tokens and the set of time stamps. For example, using CPN Tools, the tokens in a timed place represented by $2`(4,3)@5, 6 + + + 1`(2,3)@3 + + + 3`(5,5)@0$ are expressed in TIMSPATLib as:

$$2'(4,3) + 1'(2,3) + 3'(5,5);$$
$$5, 6 + 3 + 3@0;$$

A timed marking M is defined as a triple (M_u, TS) which consists of the untimed marking M_u, the time set TS, a set of time values called time stamps, $TS = \mathbb{R}_0^+$. The initial timed marking M_0 represents the initial state of the system.

The $TCPN_{SCH}$ definition given in Def. 1 considers timed transitions only, where transitions are associated with a delay $D(t)$ interpreted as the duration of the activity modeled in the event. A transition delay can correspond to machine processing or transportation time in a manufacturing environment. The delay uses the holding duration concept described in [57] for modeling the performance optimization of scheduling problems. In this concept, a timed transition is fired instantaneously but the output tokens will not be available for other transitions until the delay has elapsed. This makes the transition behave as an operation with start and release times.

The $TCPN_{SCH}$ definition introduces an important feature called *static* place which can be very useful when exploring the state space.

Definition 3. *A static place p_f is an untimed place with a static color set that does not change during the evolution of the system. The token colors are never affected by transition firing. For a place to be considered static in a TCPN, it must have two directed arcs (input and output) such that when connected to any transition in the TCPN, the input and output arc expression is the same i.e. $\forall A(a_1, a_2)\ E(a_1) = E(a_2)$ where $a_1 \in (p_f \times t)$ and $a_2 \in (t \times p_f)$, $t \in T$.*

The standard CPN formalism allows one to add as many colors as required by the model for simulating a system, be it static or dynamic information. While this is suitable for simulation purposes, it seems impractical for state space construction particularly in the case of static data. These data are commonly found in the problem definition of most manufacturing and transport systems. Examples are: deterministic processing times, transportation times, AGV routing information, etc.

In the state space exploration of CPN models, all the information required to enable or fire the transitions must be kept in the marking. Unfortunately, static data become redundant since they are propagated from one marking to the other in the state space. The place information is carried over in the marking description and repeated in every reachable state in the state space. As a consequence, the state space can get blown up as quickly as possible, leading to a premature explosion. In TIMSPAT, the static place tokens are stored once in a fixed memory location.

The static place offers a kind of flexibility to users if the information is too large to be written as an *if-then-else* expression. The command *if-then-else* can be used in some cases, however, it cannot completely replace the static place. For example, it is difficult to model alternative routings with *if-then-else* when the firing of a transition is expected to generate more than one successor.

2.1.1. TIMSPATLIb TCPN structure

The input definition files used to specify the $TCPN_{SCH}$ structure consist of a main file MDF (main.txt) and a set of N transition files TDF (transition1.txt,...,transitionN.txt). Each transition in the $TCPN_{SCH}$ is written into a separate file. The MDF specifies the initial marking M_0 (M_u and TS), the color set, the goal marking definition M_g, the time information required for heuristic evaluation (shared with the search algorithm module), constants (optional), and functions (optional) used in the transition files. Each TDF lists the arc expressions of places (both input and output where applicable) acting on a transition, guard expressions and time delay. TIMSPATLib assumes the following for TCPN modeling:

1. Place identifier: Place labels are numeric and sequential.
2. Data types: The domain of the data types are restricted to two numeric types only: integer (including large integer) and real data types. Restrictions are placed on color domains and set to facilitate an efficient exploration of the state space. As such, neither color variable and token type declaration nor initialization is required by the user. The real type is fixed at a precision of 3 decimal places using a built-in function *radiusdp*() to differentiate an integer computation from a real one. Complex data types like strings, and Boolean among others are not suitable to optimize the marking storage. These variables can be expressed numerically since most models are usually accompanied by an interpretation of the places and transitions including the meaning of the colors and data types. For instance, if a token color must take the string values 'heavy', 'medium', and 'light', the three possible values can be represented as 1, 2, and 3, respectively. Same applies to Boolean colors.
3. Color set: The library uses an *n-tuple* definition for token description; a *product* color set of integer and/or real where $n \geq 2$ or a *simple* integer or real where $n = 1$. It is only necessary to define n, a non-negative integer which describes the number of colors that will reside in a place. List, union and enumeration color sets are not supported for state space analysis.
4. Input arc expressions: The tokens used for input arc expressions are restricted to an explicit definition of free variables only. TIMSPATLib does not allow numeric values, conditional expressions or computations to be specified for input arc expressions. Also, the variables must be unique on all input arcs. Numeric values or equivalent variables intended to be used on input arc expressions can be expressed as guards. This assumption allows quick evaluation of transition bindings [33].
5. Free variables are local to a transition file and can be reused by other token colors in the other files without prior declaration.

Due to space considerations, we cannot provide a detailed explanation of the syntax description. Interested readers can consult the user guide on the tool's website [5].

2.2. Simulator – TCPN Execution

The execution of $TCPN_{SCH}$ is controlled by the simulator module. It involves the enabling and firing of transitions according to the input arc expressions, preconditions (guards) and estimated duration. Also, TIMSPAT includes the goal marking check in the simulator.

The simulator uses the TIMSPAT's net structure to evaluate markings using an event-driven approach [6, 57] for the generation of successors (See Section 2.3). Each time it receives a marking from the search

algorithm, it checks whether or not the transitions of the TCPN can be enabled given the guard conditions and input arc expressions. Once the enabled markings are fired, the simulator determines which marking has reached the goal given the TIMSPATLib M_g syntax. It then sends the reachable markings as successors with a goal marking header to the search algorithm for further evaluation. The interaction between the three TIMSPAT components is based on a continuous evaluation of markings until the search algorithm terminates the exploration. The steps of the simulator algorithm are given as follows:

1. Get a new marking from the search algorithm module.
2. For each transition in the TCPN:
   (a) Preprocessing: check if the number of tokens in each place can be reduced by evaluating the guards with at most two variables.
   (b) If one of the input places is empty or the number of tokens is less than the cardinality or the multiset of tokens in the input arc expression, exit.
   (c) Generate all the possible combinations of tokens (subsets) for all the input places.
   (d) For each token combination subset:
      i. Bind the colors of each token to their variables.
      ii. Check to ascertain whether or not it can enable the transition by evaluating the entire guard expression.
      iii. If the guard evaluates to true, go to Step 2e, else go to Step 2d.
   (e) Fire the transition with the binding:
      i. Remove the tokens from the input places of the marking and store their timestamps if color set of the place is timed.
      ii. Calculate the enabling time tau_k by taking the maximum of all token time stamps in the enabled token subset.
      iii. Calculate the firing time by adding the transition delay d to tau_k, $(tk + d)$.
      iv. Generate a new marking by adding new tokens to the output places by evaluating the output arc expressions and attaching the computed time stamp (firing time) to each token in a timed place.
      v. If the new marking is a goal marking, mark as goal.
3. Send the computed successors to the HS module.

The enabling of transitions in a CPN is usually quite expensive [69]. It has been a subject of much research in [23, 24, 33, 45]. The simulator must first compute the set of all possible bindings B for a transition t, denoted as $B(t)$ [33]. A binding b of a transition t, $b \in B(t)$ assigns a value $b(v)$ to each variable v of the transition t. It binds the tokens in the input places of transitions to the input arc expressions and guards. The variables of a transition t, denoted as $Var(t) \in V$, consist of the free variables specified in the guard and in any of the arc expressions of any arcs connected to the transition t. For example, the variables of transition t_1 in Fig. 3 is $Var(t_1) = \{j, op, m, j1, m1, opp, t\}$. A transition is enabled if the input places contain the multiset of tokens specified and the guard of the binding $G(t)\langle b\rangle$ is true.

For simulation-based performance analysis, only one successor needs to be computed in which an enabled transition and corresponding binding can be chosen based on priority, random or fair selection. To reduce the combinatorial effects of transition bindings in TIMSPATLib, we adopt the following rules: 1. The token multiset of input arc expressions must be limited to two for input places with a large number of tokens, 2. Before evaluating the bindings, the simulator first removes ineligible tokens from the input places with guards having at most two variables. This is done to reduce the number of tokens to use in the combinatorial process, and 3. When a static place is used, there must be sufficient guard conditions to trim down the number of tokens in the place. If this is not possible, an *if-then-else* operator is recommended.

An enabled transition may fire. Firing means that the tokens are removed from the input places and added to the output places of the firing transitions. In a TCPN, a transition t is time-enabled at time τ_k in a marking M denoted by $M[t\rangle_{\tau_k}$ if all the tokens to be consumed from the input places have a time stamp not later than time τ_k [64]. If a transition t fires at time τ_k, it changes M to a new marking M' denoted by $M[t\rangle_{\tau_k}M'$. M' is said to be reachable from M. In $TCPN_{SCH}$, a transition delay applies to all output tokens created at transition firing. The time stamp of a token is defined at its generation time. Firing a transition t at time τ_k with a delay d, time stamps the output tokens with the time value $\tau_k + d$.

2.3. Timed state space exploration

The performance analysis of TCPN using the RG involves the generation of a timed state space (TSS) and the traversal of the state space with a search algorithm. A TSS can be defined as a reachability set $R(TCPN, M_0)$ that comprises the set of all possible markings reachable from M_0 which minimizes a given objective function. The TSS is represented as a directed graph $TSS = (N, E, M_0)$ where N is the set of nodes and E is the set of directed edges $E = \{(M, t, M')_{\tau_k} \in N \times (T \times \mathbb{R}) \times N | M[t\rangle_{\tau_k} M'\}$. A node contains a reachable marking M including the parent identifier and any other information required by the search algorithm. A marking $M' \in V$ is a successor of (or reachable from) marking $M \in V$ if $(M, t, M')_{\tau_k} \in E$. The edges represent the transition bindings used to generate the successor marking. A path between two markings M_0 and M_g is a sequence of markings $\sigma = M_0[t_0\rangle, M_1[t_1\rangle, \ldots, M_{n-1}[t_{n-1}\rangle, M_g \in \nabla$ connected by a sequence of edges with enabling times such that $\forall i \in [0, n-1]$, $(M_i, t_i, M_{i+1}) \in E$. σ corresponds to a schedule, and ∇ is a space of feasible schedules.

Lakos & Petrucci [39] identify two different approaches to TSS generation based on their firing rules: the conservative and the optimistic approach. The conservative approach called the reduced earliest time state space (RSS) is the TSS generation method adopted by CPN Tools. It uses the eagerness-to-fire property [25] based on the global clock synchronization such that a transition is only allowed to fire if $\tau_k \leq \mathfrak{r}^*$. As a consequence, the firing of a transition t' enabled at $\tau'_k > \mathfrak{r}^*$ for a particular operation is delayed until the global clock $\mathfrak{r}^*$ advances to τ'_k or to a much later time than τ'_k depending on the prior firing sequences. When used for the optimization of inherent asynchronous systems which exhibit a high level of concurrency and parallelism [57], this property may preclude the generation of firing sequences that would lead to an optimal schedule. Since there are activities that can be performed concurrently, delaying the execution of the operation to a later time can have a negative impact on the overall performance of the system. RSS is defined as a tuple $RSS = (N, E, M_0)$ where $E = \{(M, t, M')_{\tau_k} \in N \times (T \times \mathbb{R}) \times N | M[t\rangle_{\tau_k} M', \nexists t', \tau_{k'} < \tau_k : M[t'\rangle_{\tau_{k'}}\}$

The optimistic approach called the earliest time state space (ESS) allows the firing of transitions as soon as they are enabled i.e. it includes the transition firings with $\tau_k > \mathfrak{r}^*$ in addition to those of the RSS without the global clock constraint. As a result, the firing of transitions no longer depends on the behavior of the global clock, hence, leading to an event-driven approach. ESS is a tuple $ESS = (N, E, M_0)$ where $E = \{(M, t, M')_{\tau_k} \in N \times (T \times \mathbb{R}) \times N | M[t\rangle_{\tau_k} M', \nexists \tau_{k'} < \tau_k : M[t\rangle_{\tau_{k'}}\}$. Piera & Music [57] investigated the use of the two approaches for production scheduling, highlighting the shortcomings of RSS for optimization.

ESS can be explored either in a classical manner, by evaluating both the untimed marking and time stamp together as one set for comparison in duplicate detection [8] or in a compact form, as a condensed state space (CSS) [9, 16]. In the CSS, nodes are represented as state classes using the notion of untimed marking equivalence. TIMSPAT uses the CESS as a reduced size of ESS for detecting duplicate untimed markings. During the search process, it excludes the time stamps from the duplicate detection process to avoid exploring a large state space containing several markings with the same untimed marking but different time stamp set. The CSS was originally proposed by Christensen et al. [16] using the timed equivalence (TE) method. Unlike TE, the absolute values of time stamps are not replaced with relative values. Rather, the time stamps are used for calculating the firing times of transitions and the creation times of new tokens, and to evaluate performance with respect to the objective function. TIMSPAT implements both the ESS and its condensed version (CESS) in the HS algorithms. The CESS justifies the separation of the timed marking into untimed marking and time stamp set in the descriptor used in Section 2.1. For large state space graphs, it is impractical to keep the time stamp set of all equivalent untimed markings in a class. An additional measure is required to select the most promising time stamp set to be used for exploration [6].

2.3.1. Heuristic search algorithms for TSS

A classical HS algorithm like A* [60] can be used to construct the TSS of a TCPN. A* is a best-first search that searches through the TSS by systematically expanding the most promising marking one at a time in order to find the shortest path from M_0 to M_g. It guarantees that the first solution obtained is optimal when all the markings with cost less than the optimal goal marking cost have been expanded. The search is guided by an evaluation function $f(M) = g(M) + h(M)$ that determines the cost of each marking in the search space. Cost function $g(M)$ is the actual cost to reach a marking M from M_0 and $h(M)$ is a heuristic

Algorithm 1 A* search with TCPN execution

Require: $TCPN$, M_0, M_g
1: $d_m \leftarrow 0$
2: $g(M_0) \leftarrow 0$, $f(M_0) \leftarrow h(M_0)$
3: $OPEN \leftarrow \{M_0\}$, $CLOSED \leftarrow \{M_0, d_m\}$
4: **while** $OPEN \neq \emptyset$ **do**
5: $M \leftarrow OPEN \setminus \{M_{best}\}$
6: **if** $IsM_g(M)$ **then**
7: $M_f \leftarrow M'$, construct firing sequence
exit
8: **else**
9: **for all** enabled transitions $t \in T : M[t\rangle_{\tau_k} M', \nexists \tau_{k'} < \tau_k : M[t\rangle_{\tau_{k'}}\}$ **do**
10: **if** $M'(M'_u) \notin CLOSED$ **then**
11: $CLOSED \leftarrow CLOSED \cup \{M', i+1\}$
12: $OPEN[i+1] \leftarrow OPEN[i+1] \cup \{M'\}$
13: **else**
14: $CSS((f(M_{stored}), g(M_{stored})), (f(M'), g(M')))$
15: **return** M_f and firing sequence (solution path)

function that estimates the remaining cost to reach M_g from M. A* guarantees that the search always finds an optimal solution if $h(M)$ is admissible i.e. it is a lower bound that does not overestimate the cost to goal, $h(M) \leq \mathrm{h}^*(M)$, $\forall M$ where $\mathrm{h}^*(M)$ is the cost of the optimal path from M to M_g.

Like A*, most HS algorithms employ two data structures: open list and closed list. The open list (OPEN) is a queue that stores the markings that have been generated but not yet expanded, whereas the closed list (CLOSED) which is usually represented by a hash table, stores the already-expanded (visited) markings. The heuristic search algorithm determines how OPEN is implemented, as a priority or non-priority queue. A* uses a priority OPEN in which markings are sorted in the increasing values of $f(M)$. Contrary to the standard approach, TIMSPAT implements CLOSED as a list that keeps both the open and closed markings. This is due to the high run-time cost incurred on performing duplicate detection on a queue. To avoid duplicating markings on both lists, TIMSPAT keeps only the pointers to the open markings in OPEN and their corresponding heuristic cost values. The common data structure allows TIMSPAT to integrate different heuristic search algorithms in the tool.

The pseudocode for the A* search combined with TCPN execution is given in Algorithm 1. Here, the algorithm uses both $f(M)$ and $g(M)$ for the CSS duplicate detection procedure $CSS((f(M_{stored}), g(M_{stored})), (f(M'), g(M')))$ to provide a more accurate estimate in selecting the most promising time stamp set [7].

A* offers completeness and optimality guarantee. However, for sizable problems, it might require a large amount of search space and computational time effort before an optimal solution can be reached [9]. Besides A*, eight other algorithms have been implemented: 1. Breadth-first iterative deepening A* search (BFIDA*) [9, 76], 2. BFIDA* with scalable layered duplicate detection (BFIDA*-SLDD) [8], 3. Beam A* search (BAS) [49], 4. A* with backtracking (A*-BT) [31], 5. Dynamic window search (DWS) [50, 59], 6. Anytime layered search (ALS) [9], 7. Anytime column adaptive search (ACAS) [6], and 8. Depth-first branch and bound (DFBnB) [47, 74, 75]. Each algorithm is classified according to the space/time tradeoff criterion as space-efficient (SE), time-efficient (TE) or space/time-efficient (STE). ESS and CESS form the base classes of the heuristic search algorithms in TIMSPAT despite the fact that BAS and DWS selects only a subset of successors generated at each marking. The details of each algorithm can be found in its respective citation.

For an algorithm to be considered SE, the reduction of the memory requirements must not affect the optimality of the schedule i.e. the heuristic function must be admissible and no inadmissible pruning technique should be adopted. These algorithms are oriented toward obtaining optimal solutions if given sufficient time, in addition to an efficient use of memory. Upper bounds are used to remove paths whose markings will not lead to a better solution. The SE algorithms do not terminate the search until the optimal solution is found. BFIDA* and BFIDA*-SLDD fall under the SE class. Like A*, these algorithms are oriented towards obtaining

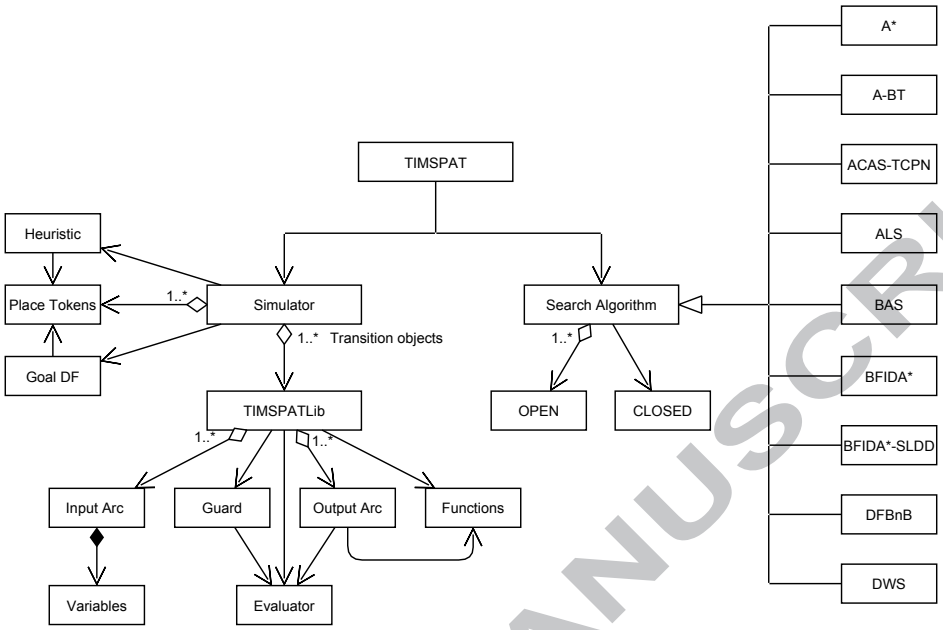


Figure 2: Relationship between the three components and the classes used in TIMSPAT.

optimal solutions.

For time efficiency, the only criterion is that the algorithm returns a solution (either optimal or near-optimal) in a reasonable amount of time irrespective of the type of heuristic function and the pruning technique employed. The TE class includes hybrid HS (HHS) algorithms like BAS, A*-BT, and DWS. These algorithms use predefined pruning rules to limit the memory consumption of the search space and to find feasible solutions in a reasonable amount of time. Basically, they limit the frequent backtracking of the A* search to prevent the search space from degenerating into a breadth search. A controlled deepening search is usually favored to drive the search towards a suboptimal solution quickly. Here, optimality is sacrificed for computation time and memory reduction. The TE algorithms terminate the search as soon as the first solution is obtained.

STE algorithms must meet the SE requirements in addition to returning solution at a reasonable amount of time. Basically, they consist of anytime algorithms that report solution at different time intervals and are guaranteed to provide the best solution obtained so far whenever interrupted. STE algorithms can be considered as a special class of HHS methods. Examples are: ALS, ACAS, and DFBnB. They do not stop the search at the first solution. Instead, the solution is continuously improved over time until the search obtains the optimal solution provided the available memory is sufficient to guarantee optimality. The algorithms trade off solution quality and computational time. The incumbent best solution is used as an upper bound to restrict the number of generated successors and to periodically prune markings that will not lead to a better solution.

Fig. 2 shows the interaction between the three components and the relationships between the classes used in TIMSPAT. The evaluator class is used to evaluate and compute guard conditions, functions, and other mathematical expressions in the output arc.

2.3.2. Heuristic functions

Three admissible heuristic functions are commonly used in PNHS approach. The first one sets $h_1(M) = 0$ assuming no heuristic information is available. This is suitable for manufacturing systems with routing flexibilities or alternative routings and in cases where the run-time overhead for heuristic computation is quite high. However, the resulting lower bound $f(M)$ might be too weak to reach an optimal schedule in a reasonable time. On the other hand, it can be very useful in cases where the algorithms are designed to

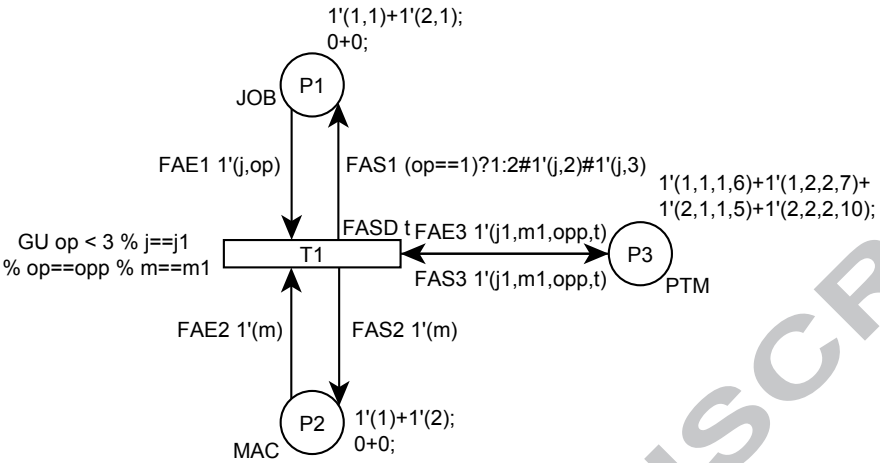


Figure 3: TCPN of a 2×2 job shop instance.

return suboptimal solutions quickly [9]. The second one is called the job heuristic function [41, 43]: $h_2(M) = max_k\{\xi_k(M), k = 1, 2, \ldots, N\}$ where $\xi_k(M)$ is the sum of operation times on uncompleted processing stages to be undergone by each kth job when the current system marking is represented by M, and N is the total number of jobs. The third is called the machine heuristic function [71]: $h_3(M) = max_i\{\xi_i(M), i = 1, 2, \ldots, R\}$ where $\xi_i(M)$ is the sum of operation times of those remaining operations for all jobs which are planned to be processed on the ith resource, and R is the total number of resources. These three functions have been used in [6, 8, 9].

In the formulations of $h_2(M)$ and $h_3(M)$, the timed marking information is not used in the computation. To this effect, Li et al. [43] propose tighter lower bound estimates for TPN that consider the earliest available time of machines and jobs based on the information from the timed marking. The individual time stamps of tokens are used to calculate the lower bound. The equivalent expression $f_{2m}(M)$ for a TCPN (Fig. 3 as an example) proposed in [43] is as follows: Given a token s_j of a job J_k in place p_n of color set JOB containing color variables j(job identifier) and op(operation identifier) to be processed on a list of machine tokens s_{m_i} in place p_m, where $i = \{op_c, op_c + 1, \ldots, op_f\}$. m_i is the machine list to be used for the job's remaining operations starting from the current operation op_c to the final operation op_f. Then, the $f_{J_k}(M)$ of a job J_k is estimated as:

$$f_{J_k}(M) = max\left\{ts_1{}^j + \sum_{op_c \le i \le op_f} D^i\,(t,b), ts_1{}^{m_{op_c}} + \sum_{op_c \le i \le op_f} D^i\,(t,b), \right. \\ \left. ts_1{}^{m_{op_c+1}} + \sum_{op_c+1 \le i \le op_f} D^i\,(t,b), \ldots, ts_1{}^{m_{op_f}} + D^{op_f}\,(t,b)\right\} \tag{3}$$

where $max(ts_1{}^j, ts_1{}^{m_{op_c}})$ corresponds to $g_{J_k}(M)$, the earliest available time (firing time) of job J_k and $D^i\,(t,b)$ is the transition delay for bindings $b(j) = k$ and $b(op) = i$, the processing time of the job for each operation. This expression assumes a single lot size $m(s_j) = 1$ for all jobs.

The overall lower bound $f_{2m}(M)$ for $h_{2m}(M)$ is given as:

$$f_{2m}(M) = max\left\{f_{J_k}(M)\right\}, \quad k = 1, 2, \ldots, N \tag{4}$$

A similar modification is made to $f_3(M)$ where the earliest start time of the next job operation on a machine is computed using $f_{J_k}(M)$ before adding the sum of operation times. To show the effectiveness of these heuristic functions, Fig. 4 depict the A* search of the CESS graph (TCPN in Fig. 3) given in [6] using the previous $f_2(M)$ and modified $f_{2m}(M)$ job heuristic functions, while Fig. 5 shows those of the machine

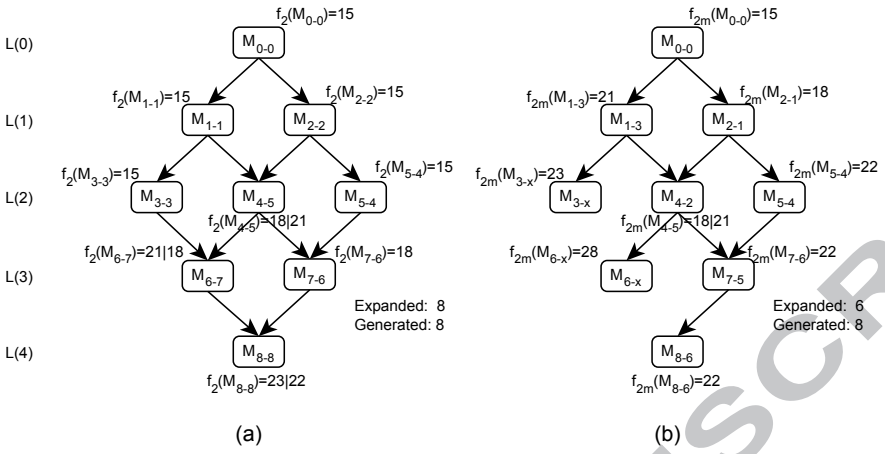


Figure 4: A* search using (a) $f_2(M)$, and (b) $f_{2m}(M)$.

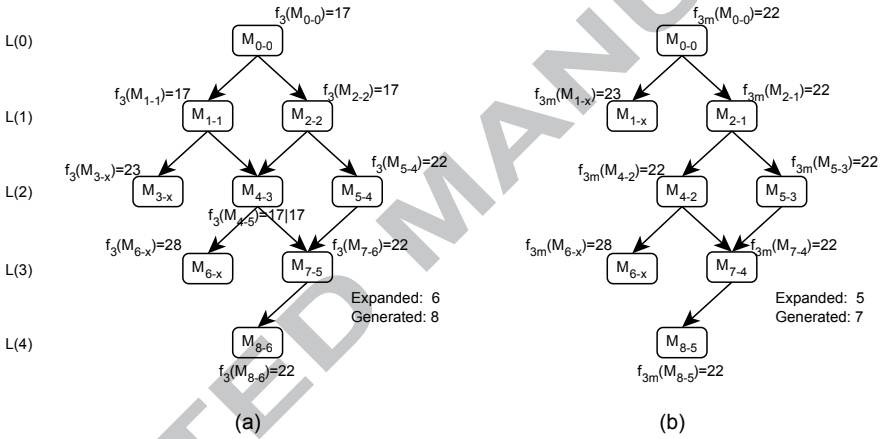


Figure 5: A* search using (a) $f_3(M)$, and (b) $f_{3m}(M)$.

heuristic functions. The x and y variables in the marking identifier M_{x-y} represent the order of expansion of the CESS by BFS and A* respectively. As observed in the graphs, the improved heuristic functions proved to be more informed than the previous ones, expanding and storing fewer markings i.e. $h_2(m) \leq h_{2m}(M)$ and $h_3(m) \leq h_{3m}(M)$. Also, Fig. 4a shows the importance of a good estimate as the A* search degenerated into a breadth-first.

While the job and machine heuristic functions can be used separately, they can also be formulated as $f(M) = max(f_{2m}(M), f_{3m}(M))$ [43] to give a more accurate lower bound. The computation may become more time consuming, especially for TCPNs. However, in systems with alternative routings in which more than one machine can be used to process the operation (with different processing times) of some jobs, only the job heuristic function and $f_1(M)$ seem applicable. Functions $f_{2m}(M)$ and $f_{3m}(M)$ are not admissible. Here, we propose a modified job heuristic function for alternative routings by replacing the processing time of each operation and the time stamp of machines with the minimum processing time and the earliest available

time, respectively.

$$f_{J_k}{}^{alt}(M) = max\Bigg\{ts_1{}^j + \sum_{op_c \le i \le op_f} min_i\delta_i, min_{s_{m_{op_c}} \in S_m}(tm[s_{m_{op_c}}]) + \sum_{op_c \le i \le op_f} min_i\delta_i,$$
$$min_{s_{m_{op_c+1}} \in S_m}(tm[s_{m_{op_c+1}}]) + \sum_{op_c+1 \le i \le op_f} min_i\delta_i, \tag{5}$$
$$\ldots, min_{s_{m_{op_f}} \in S_m}(tm[s_{m_{op_f}}]) + min_{op_f}\delta_{op_f}\Bigg\}$$

where $tm[s_{m_i}]$ is the time stamp list of machines that can be used to process a job at a given operation, and $\delta_i = \{D^1(t,b), \ldots, D^l(t,b)\}$ is the list of transition delays for l number of alternative machines that can process J_k at operation i.

Theorem 1. *$f_{2m}{}^{alt}(M) \le f^*(M)$ where $f^*(M)$ is the cost of the optimal solution from M*

Proof. Let $g_J(M) = ts_1{}^j$, the current time stamp of the job token, and $\tau_k = max(ts_1{}^j, ts^{m_{op_c}})$ represents the firing time of the enabled transition t, the minimum start time of the job processing, where $ts^{m_{op_c}} = min_{s_{m_{op_c}} \in S_m}(tm[s_{m_{op_c}}])$ is the minimum available time of the set of machines that can be used for processing. The processing will finish at the earliest time $\tau_i = \tau_k + \delta_k$ where δ_k is the minimum processing time $min_i\delta_i$ of the job assuming that it undergoes operation at the machine with the least processing time. Then, $\tau_i^* \le \tau_k{}^* + D^*(t,b)$, the time stamp of the actual selected machine $ts^{m_{op_c}*} \ge ts^{m_{op_c}}$, and $D^*(t,b)$ is the corresponding transition delay $D^*(t,b) \ge \delta_k$. The completed time at the next processing stage $i+1$ then becomes $\tau_{i+1} = max(\tau_i, ts^{m_{op_c+1}}) + \delta_{i+1}$, which will be less than or equal to $\tau_{i+1}{}^*$. The job processing is estimated to always start at the earliest available time on one of the machines and finishes with the least processing time, i.e. the minimum completion time at each stage will always be less than or equal to the actual completion time, $\tau_i \le \tau_i^*$. Therefore, $f_{J_k}{}^{alt}(M)$ guarantees that $f_{2m}{}^{alt}(M)$ is an admissible lowerbound that will always be less than or equal to $f^*(M)$.

Note that, in this expression, the time stamp of the selected machine is updated at each iteration to avoid reusing the same machine if it were to be available for more than one processing stage. The machine will be considered to have processed an operation in case it is part of another set of alternative machines to be used for subsequent operations. This ensures that the calculation of the lower bound advances without overloading a machine, and resource utilization is spread out to the other machines.

3. Case study

We consider a case study of a real flexible manufacturing cell (FMC) of an eyeglass production system [53]. The system consists of three computer numerical controlled machines $M1$, $M2$, and $M3$, an automated dual-gripper crane $R(G1, G2)$, and a conveyor. The layout of the cell is shown in Fig. 6. There are three types of eyeglasses E_1, E_2, and E_3 in which each eyeglass is composed of two lenses (left and right). Each lens must be processed separately since the machine processing time of some operations depends on the lens type. Hence, the total number of part types to be processed is six: J_1 and J_2 for E_1, J_3 and J_4 for E_2, and J_5 and J_6 for E_3.

Parts undergo two machining operations in the same sequence as in a flow shop. The first operation is performed on $M1$, and the second on either $M2$ or $M3$. Machine $M1$ is used to verify whether the lenses have the correct dimension specification, whereas machines $M2$ and $M3$ perform the same function and are used to bevel the lenses. All the part types have the same processing time of 4 s in the first operation but the beveling operation varies depending on the part type: 120 s for E_2, 540 s for E_3, and 215 s and 220 s for J_1 and J_2 of E_1 respectively. Each machine can process at most one operation at a time.

Parts arrive in buckets of a pair of lens ($B_1(E_1)$, $B_2(E_2)$, and $B_3(E_3)$) on the conveyor to the load/unload (L/U) area. Each bucket contains an eyeglass type, and the L/U area is the working area of the crane

operations on the conveyor. The crane is used to transfer the parts (in a horizontal movement) between the conveyor and the machine and back to the L/U area after the part processing is completed. However, the L/U area is constrained to three buckets. This is due to the restricted working area of the crane's gripper to load and unload parts. A slot is freed only when the two parts in a bucket are fully processed and moved out from the working area. The crane can load and unload parts at four pickup/delivery (P/D) points: 1. 0 - conveyor loading position, 2. 1 - $M1$, 3. 2 - $M2/M3$, and 4. 3 - conveyor unloading position. The movement time of the crane from one position to the other, and the loading and unloading of parts takes 4 s each. Both arms of the crane can be used to load and unload parts. The operation of the dual-gripper crane works more like a multi-load AGV that performs empty and loaded trips [7].

Each job has a total of eight operations (six handling and two machining operations): 1. Unload or pickup from bucket, 2. Transport and delivery or load to $M1$, 3. Processing in $M1$, 4. Unload from $M1$, 5. Transport and delivery to $M2$ or $M3$, 6. Processing in $M2$ or $M3$, 7. Unload from $M2$ or $M3$, and 8. Transport and delivery to bucket. The scheduling problem is formulated as follows: Given the FMC layout and the production mix, determine the starting and completion times of each part on each machine and the movement operations of the crane between machines together with the assignment according to the makespan minimization objective C_{max}.

Figure. 7 shows the TCPN model of the manufacturing cell. The TCPN has six places and seven transitions. The meaning of the places together with the color set and variables, and transitions is given in Table 2 and Table 3 respectively. The main file contains the initial marking of the production mix $E_1 = 1$, $E_2 = 1$,and $E_3 = 1$ with zero starting times. For dynamic scheduling purposes, the starting times can be changed to reflect the current state of the system.

$$\begin{aligned} 1\ 1'(1,1,2,1) + 1'(2,3,4,1) + 1'(3,5,6,1); 2\ 1'(0); 3\ ; 4\ 1'(0,0,0,0,0,0,0); 5\ 1'(1,0,1,0,0,3) \\ +1'(2,0,2,0,0,6) + 1'(3,0,2,0,0,6); 6\ ; /*\ initial\ M_u\ */ \\ 0+0+0; 0;\ ; 0; 0+0+0;\ ; /*\ initial\ time\ stamp\ set\ */ \\ \text{CS}\ WKL; CNT; BCK; CRN; MCH; OUT; /*\ place\ color\ set\ */ \\ \text{EF}\ \#; 1'(0); \#; 1'(0,0,*,0,0,0,0); 3'(*,0,*,0,0,*); 3'(*,*,*,9); /*\ goal\ marking\ */ \end{aligned}$$

A P/D request involves 5 operational sequence: the prior assignment of the crane to the part, the movement of the crane to the resource location (conveyor or machine) for pickup if its current position is different from the P/D location, the unloading of the part from the resource, the delivery of the part to the destination resource, and the loading of the part to the resource. Transitions $T2$ and $T5$ execute the first

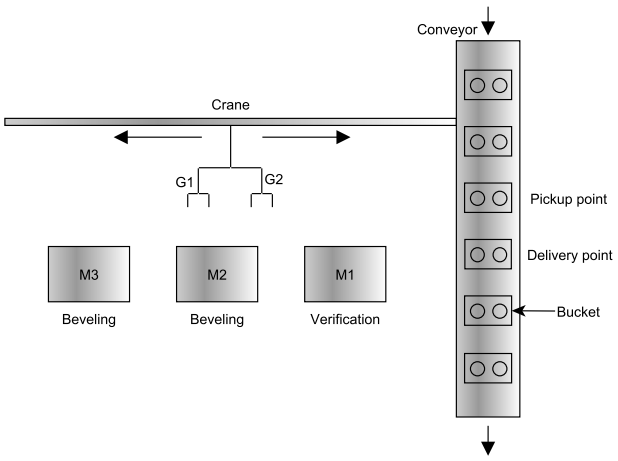


Figure 6: The layout of the flexible manufacturing cell.

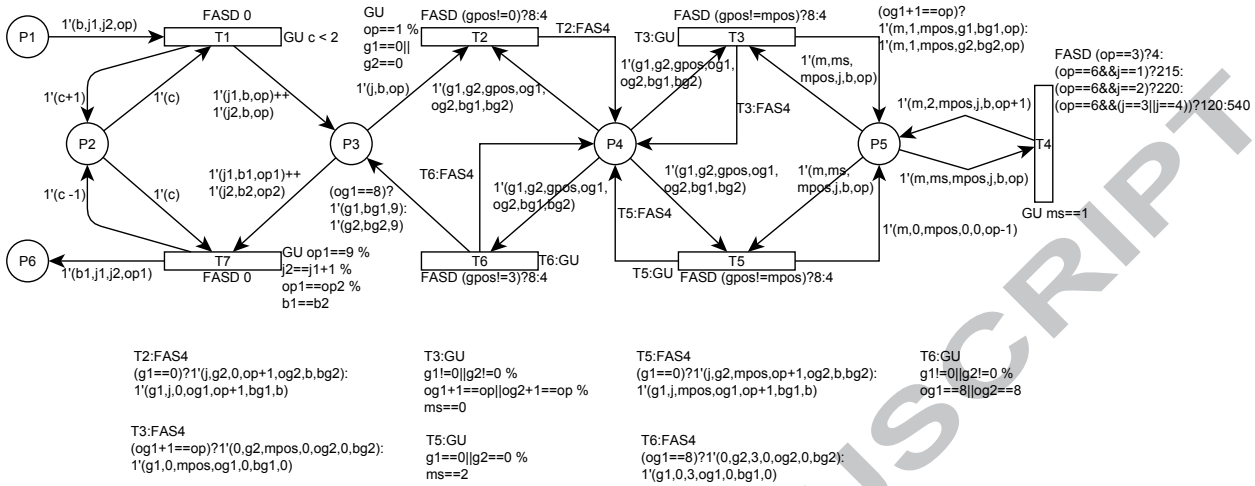


Figure 7: The TCPN model of the FMC developed using TIMSPAT syntax library.

Table 2: Interpretation of places and colors.

Place	Color set (CS)	Color(s)	Description
$P1$	$WKL = 4, timed;$	$\langle b, j1, j2, op\rangle$	Production mix workload: Left lens $j1$ and right lens $j2$ in bucket b with starting operation sequence identifier op
$P2$	$CNT = 1, timed;$	$\langle c\rangle$	Counter for the number of unprocessed buckets on the conveyor in the loading area
$P3$	$BCK = 3, timed;$	$\langle j, b, op\rangle$	Buckets on the conveyor: Unprocessed lens j in bucket b if op=1, otherwise processed if op=9
$P4$	$CRN = 7, timed;$	$\langle g1, g2, gpos, og1, og2, bg1, bg2\rangle$	Crane status: Crane at position $gpos$ with grippers $g1$ and $g2$ (free = 0, busy > 0), and operation sequence identifiers $og1, og2$ and bucket identifiers $bg1, bg2$ for each gripper. $og1, og2, bg1, bg2 > 0$ if parts (lens) are held in grippers
$P5$	$MCH = 6, timed;$	$\langle m, ms, mpos, j, b, op\rangle$	Machine status: Machine m at position $mpos$ with status ms (0 - available, 1 - busy, 2 - waiting for part to be picked up). $j, b, op > 0$ if $ms > 0$
$P6$	$OUT = 3, timed;$	$\langle b, j1, j2, op\rangle$	Processed buckets moved out from the working area

three operations concurrently for conveyor and machine pickup respectively, while $T3$ and $T6$ perform the last two operations for machine and conveyor delivery respectively. The production flow of a single part in the system gives the sequence of transition firings: $T1 \rightarrow T2 \rightarrow T3 \rightarrow T4 \rightarrow T5 \rightarrow T3 \rightarrow T4 \rightarrow T5 \rightarrow T6 \rightarrow T7$.

The TCPN model is quite different from the nets proposed by Narciso et al. [53] and Mujica & Piera [51]. In the previous nets, the movement of the crane is not properly controlled as the source or destination of a P/D request is not included in the crane's behavior. The crane can move to either of the potential P/D positions without a prior assignment for P/D request. Since the studied FMC is a bufferless system, the TCPN minimizes blocking using P/D assignment together with the status of the machines as specified in the guards of transitions $T2$ and $T5$. However, deadlock occurrence is still inevitable if the crane grippers are holding parts whose destination machines have parts that are waiting to be picked up by the crane. The deadlock-free scheduling approach used in this thesis has been treated in [9].

4. Performance evaluation and benchmarking

This section evaluates the performance of the nine algorithms on the case study. We consider 15 different production mix scenarios, shown in Table 4. The instances contain 5 small ($BGS1$–$BGS5$), 6 medium ($BGM1$–$BGM6$), and 4 large ($BGL1$–$BGL4$) workload mixes. The first large instance $BGL1$ ($30\%\ E_1, 60\%\ E_2, 10\%\ E_3$) represents a real mix for the manufacturing cell. The experiments were conducted on a 2.60GHz AMD Opteron processor PC with 4GB RAM.

Each algorithm is benchmarked against the others in its category, and the best performing in each class is then compared with those of the other classes. Since each class has its own trade off, CPU time of 3600 s and 4GB RAM limits were set for the TE and STE classes whereas only the maximum memory limit was set for the SE class. A* is not considered for comparison as it was only able to solve the small instances, even with the CESS.

4.1. SE class

The SE algorithms use the cost threshold CT as a bound to prune markings with $f(M) > CT$ in each iteration of the breadth-first branch and bound search. The algorithms start with $f(M_0)$ as the initial CT. If no goal marking is found, the search is repeated with a new CT value until a solution is reached. Successive values of CT relies on the minimum $f(M)$ (f_{min}) of the unexpanded markings in the previous iteration. However, it comes with an overhead of marking re-expansion each time the search is restarted. If the increments are too small to reach $f(M_g)$ in a reasonable amount of time, it may result in a large number of iterations. To circumvent this problem, [65] propose to double CT after each iteration for the classical IDA*. This measure can degenerate to a breadth-first search. To achieve a good space-number of iteration trade-off, we computed successive CTs as $CT = max(f_{min}, ubf * f_{min}), \forall ubf \in \{1.0, 2.0\}$ where ubf is a multiplier. We experimented with different values of ubf to determine a good factor for the instances. From the preliminary results obtained, a ubf value of 1.4 achieves a good trade-off.

Table 5 shows the scheduling results of the two SE algorithms on the 15 instances. The two are practically the same. They expand the same of number of markings (EM) and have an almost equal CPU time but differ in the number of stored markings as shown in the *Peak* column. BFIDA*-SLDD leverages the regular structures found in manufacturing systems to reduce the number of stored markings. The *Depth* is the total number of operations or fired transitions required to reach the optimal M_g from M_0, where each operation corresponds to a level in the state space.

As observed on the result table, the ubf of 1.4 significantly reduces the number of search iterations across all instances. As a result, the computational times were also reduced for the small and medium instances. For example, the BFIDA* search of $BGS1$ using the standard increment of $ubf = 1$ solved the instance in 156 iterations with a CPU time of 3753 s and EM of 9.6×10^6. As the instance size becomes larger, the standard BFIDA* ran out of memory (o.o.m) starting from the $BGM3$ instance due to the exponential increase of the state space size. BFIDA*-SLDD solved all the instances with a minimal amount of memory space. It used less than 2GB RAM for the largest instance that would require over 200GB RAM if the entire state space were to be stored in memory. The CPU times can be considered reasonable up to instance $BGL1$ considering the fact that it is an optimal algorithm. However, it took over 38 h, 64 h, and h to solve $BGL2$,

Table 3: Transitions and their meanings

Transition	Description
$T1$	Conveyor move in of unprocessed buckets in the L/U area
$T2$	Unloading of parts from bucket for P/D including the empty move of the crane if applicable
$T3$	Movement of crane to the destination machine for delivery including the loading of parts
$T4$	Processing of parts in machines
$T5$	Movement of crane to the destination machine for pickup including the unloading of parts
$T6$	Final delivery of processed parts to bucket in the conveyor's L/U area including unloading of parts
$T7$	Conveyor moves out processed buckets

Table 4: Production mix instances for the three eyeglass types.

Instance	Number of eyeglass types			Number of parts
	E_1	E_2	E_3	
BGS1	1	1	1	6
BGS2	2	2	0	8
BGS3	4	0	0	8
BGS4	4	2	1	14
BGS5	3	3	3	18
BGM1	2	5	3	20
BGM2	7	4	2	26
BGM3	4	7	4	30
BGM4	4	8	5	34
BGM5	7	5	5	34
BGM6	6	10	4	40
BGL1	7	13	2	44
BGL2	10	17	5	64
BGL3	15	20	7	84
BGL4	20	20	20	120

Table 5: Scheduling results of SE algorithms.

Instance	State space			BFIDA*		BFIDA*-SLDD				DFS-CSS [53]	
	Itr	Depth	EM	Peak	CPU (s)	Peak	CPU (s)	Reduction (%)	C_{max}	C_{best}	CPU (s)
BGS1	4	54	150,548	114,371	59.8	4,869	59.1	95.74	975	1027	45077
BGS2	5	72	171,140	117,869	71.9	4,932	71.4	95.82	783	911	54207
BGS3	3	72	19,905	18,297	7.9	496	7.9	97.29	978	1111	2978
BGS4	5	126	1,608,257	1,014,911	718.2	21,237	715.8	97.91	1794	---	---
BGS5	5	162	4,735,124	3,000,133	2216.9	57,783	2211.7	98.07	2802	---	---
BGM1	5	180	5,211,153	3,363,369	2436.2	55,308	2431.2	98.36	2835	---	---
BGM2	5	234	11,718,087	6,104,427	5663.3	73,095	5583.1	98.80	3301	---	---
BGM3	5	270	29,217,707	---	---	131,865	14463.4		4110	---	---
BGM4	5	306	32,719,717	---	---	163,967	16060.0		4794	---	---
BGM5	5	306	32,228,078	---	---	186,191	16056.4		5089	---	---
BGM6	6	360	52,626,061	---	---	201,008	26330.5		4965	---	---
BGL1	5	396	30,251,710	---	---	125,448	15030.2		4489	6790	513540
BGL2	6	576	179,031,180	---	---	406,956	93592.5		7359	---	---
BGL3	5	756	274,552,404	---	---	820,818	159148.0		10009	---	---
BGL4	4	1080	1,061,683,289	---	---	2,295,016	649915.0		18330	---	---

Itr - Number of iterations

BGL3, and *BGL4* respectively. When compared with the depth-first search (DFS-CSS) algorithm proposed by [53] where some instances coincide, it is a significant improvement in terms of both CPU time and best makespan (C_{best}) obtained.

4.2. TE class

Each algorithm in the TE class has its own input parameter settings, used as a measure to reduce the memory and computational time complexity. However, they share a common parameter that determines a priori the number of markings to be stored. Since increasing the input values do not guarantee a high solution quality, different values must be experimented to achieve a good solution quality-time trade-off.

The A*-BT algorithm requires only one input parameter called threshold, M_{max} which is used to control the maximum number of markings stored in OPEN. It starts exploring the state space in a best-first order using A*. Each time M_{max} is reached, it creates a new search region or level by initiating a backtracking to the previous level where the best marking in OPEN is used as a root node for another A* search until a solution is found.

BAS requires two inputs: 1. beam width *bw*, and 2. cutoff *co*. The beam width is used to limit the number of markings expanded at each depth of the state space whereas cutoff limits the size of OPEN to a

Table 6: Scheduling results of TE algorithms.

Instance	A*-BT					BAS					DWS				
	M_{max}	C_{best}	CPU	RPD_C	RPD_{CPU}	bw	C_{best}	CPU	RPD_C	RPD_{CPU}	$maxz$	C_{best}	CPU	RPD_C	RPD_{CPU}
BGS1	1000	1012	2.7	3.8	41.2	100	980	1.9	0.5	0.0	200	975*	3.9	0.0	106.0
BGS2	100	831	0.2	5.1	0.0	200	791*	5.2	0.0	2026.0	100	799	2.8	1.0	1072.3
BGS3	1000	1015	5.8	3.8	0.0	500	983	7.4	0.5	27.3	500	978*	12.2	0.0	110.0
BGS4	1000	1889	9.0	4.4	0.0	500	1822	25.2	0.7	180.3	1000	1809*	43.9	0.0	388.1
BGS5	100	2989	0.7	4.0	0.0	1000	2873*	68.8	0.0	9826.0	1000	2877	60.7	0.1	8665.2
BGM1	100	3079	0.8	3.8	0.0	500	2966*	39.2	0.0	5012.4	50	2974	4.7	0.3	509.5
BGM2	200	3580	1.9	2.1	0.0	200	3538	21.5	0.9	1032.3	50	3506*	8.1	0.0	326.9
BGM3	1000	5142	11.7	22.1	81.6	50	4212*	6.5	0.0	0.0	50	4270	49.4	1.4	664.0
BGM4	10	5172	0.2	3.1	0.0	10	5018*	1.6	0.0	561.9	50	5086	8.4	1.4	3321.0
BGM5	100	5527	1.6	3.5	0.0	10	5342*	1.6	0.0	1.6	100	5367	15.0	0.5	866.9
BGM6	10	5498	0.3	5.2	0.0	1000	5228*	170.2	0.0	50897.9	200	5396	33.6	3.2	9972.3
BGL1	10	4803*	0.3	2.3	0.0	200	4895	39.2	4.2	11970.7	500	4696	84.8	0.0	25967.5
BGL2	10	7954	0.5	0.5	0.0	10	7917*	3.2	0.0	486.5	10	7921	6.3	0.1	1065.6
BGL3	10	10507*	0.7	0.0	0.0	10	10803	4.3	2.8	499.3	50	11123	27.7	5.9	3760.3
BGL4	10	19215*	1.1	0.0	0.0	10	19306	6.5	0.5	504.7	50	19984	36.2	4.0	3285.4
Av. RPD				4.24	8.19				0.68	5535.12				1.18	4005.39

*-Best solution

certain value to avoid an exponential growth. On the other hand, DWS restricts the state space to a dynamic search window between a minimum depth, bottom-depth bd and a maximum depth, top-depth td. For the search window to advance, DWS constrains the number of markings generated at td to a certain value called max-top $maxt$. Once $maxt$ is exceeded, the values of bd and td are increased by one. To avoid exponential growth, DWS also keeps the most promising markings to be explored at each depth of the search window to a fixed size called max-size $maxz$. If a level becomes full (i.e. $maxz$ has been reached), a new marking M is added to the level only if there is a stored marking M_s with $f(M_s) > f(M)$. As such, three input values $(td, maxt, maxz)$ are required for DWS to run. The bottom depth starts from zero and [50] propose to set $maxt = maxz$.

To provide a fair comparison, we experimented the following values for the three algorithms: $M_{max} \in \{10, 50, 100, 200, 500, 1000\}$, $(bw, co) \in \{(10, 150), (50, 750), (100, 1500), (200, 3000), (500, 7500), (1000, 15000)\}$ and $(td, maxz) \in \{(30, 10), (30, 50), (30, 100), (30, 200), (30, 500), (30, 1000)\}$. Since we are not restricting the maximum number of successors to be generated at each marking, the input values must be large enough to reach a goal marking. For the DWS, the proposed initial $td = 15$ did not generate a feasible solution for most of the instances.

Table 6 presents the results obtained by the three algorithms for the 15 instances. For each algorithm, we show the input parameter value that returned the best solution taking into account the CPU time and compare the relative percentage deviation (RPD) from the best solution (RPD_C) and CPU time (RPD_{CPU}) between the three algorithms. The A*-BT algorithm provided the best solution-time quality trade-off considering the little computation time used to obtain the first solutions. Unlike the other algorithms, the CPU time is somewhat maintained across all instances without exceeding 12 s. In terms of solution quality, BAS is superior using the average deviation, albeit with a higher runtime overhead. Also, BAS proved to be more time-efficient than the others in 8 of the instances.

4.3. STE class

Like the TE class, the STE algorithms also define some input parameter settings with the exception of DFBnB. In ACAS, three parameters are needed before exploration: the initial column width ω, the column width increment α, and the maximum column width ω_{max}. ACAS is an adaptive search that focuses on improving the current best solution obtained in a minimal time whenever possible. It increases ω by α if the solution is not improved after a certain number of iterations. The width increment is stopped when ω reaches ω_{max} to avoid unnecessary memory usage if the solution cannot be improved within the time frame. Once the solution is improved, the column width ω is reset to its initial value. On the other hand, ALS does not require an input parameter a priori. However, to return solutions in a good time frame for problems

Table 7: Scheduling results of STE algorithms.

Instance	ACAS-TCPN					ALS					DFBnB				
	First solution		Best solution (CPU=3600)			First solution		Best solution (CPU=3600)			First solution		Best solution (CPU=3600)		
	C_f	CPU_f	C_{best}	CPU_{best}	RPD_C	C_f	CPU_f	C_{best}	CPU_{best}	RPD_C	C_f	CPU_f	C_{best}	CPU_{best}	RPD_C
BGS1	999	0.09	**975**	88.4	0.0	999	0.09	**975**	94.8	0.0	992	0.03	**975**	3476.1	0.0
BGS2	863	0.13	**783**	54.2	0.0	863	0.13	**783**	93.7	0.0	843	0.03	787	309.8	0.5
BGS3	1051	0.14	**978**	13.8	0.0	1051	0.17	**978**	19.4	0.0	1038	0.03	**978**	1881.3	0.0
BGS4	1846	0.23	**1794**	704.7	0.0	1846	0.25	**1794**	752.2	0.0	1820	0.06	1810	17.6	0.9
BGS5	2938	0.31	2802	510.0	0.0	2938	0.31	2802	1220.4	0.0	2858	0.08	2858	0.1	2.0
BGM1	2947	0.34	2835	2513.8	0.0	2947	0.34	2835	778.3	0.0	2895	0.08	2895	0.1	2.1
BGM2	3485	1.67	3324	3036.2	0.3	3392	0.59	3316	3045.4	0.1	3332	0.11	3313	223.8	0.0
BGM3	4748	0.73	4127	2332.4	0.0	4217	0.53	4135	1891.4	0.2	4170	0.13	4166	0.2	0.9
BGM4	4901	0.61	4804	2291.0	0.0	4901	0.61	4809	1866.9	0.1	4854	0.14	4850	0.2	1.0
BGM5	5161	2.34	5114	1123.1	0.2	5199	1.09	5119	1302.3	0.3	5120	0.14	5106	235.7	0.0
BGM6	5226	4.77	5000	2345.7	0.2	5114	1.20	4997	67.6	0.2	4996	0.17	4988	4.5	0.0
BGL1	4713	5.32	4544	109.8	0.7	4597	1.30	4519	2240.0	0.2	4520	0.19	4512	4.9	0.0
BGL2	7570	8.60	7430	141.4	0.5	7497	10.76	7424	1851.6	0.5	7400	0.28	7390	55.6	0.0
BGL3	10420	24.01	10099	3572.5	0.2	10334	74.99	10148	399.5	0.7	10094	0.37	10082	57.0	0.0
BGL4	19043	30.67	18645	1989.7	1.2	18919	143.51	18506	1671.3	0.5	18430	0.58	18418	66.3	0.0
Av. RPD					0.23					0.17					0.49

Bold solution - Converged to optimal

with large branching factor (number of successors), it is advised to limit the number of markings expanded at each level called the expansion width $e\omega$. We set $\omega = e\omega = 5$, $\alpha = 5$, and $\omega_{max} = 50$ [6].

The three algorithms differ in two aspects. The first is the number of markings expanded at each level in an iteration. ACAS and ALS is determined by ω and $e\omega$ respectively, whereas DFBnB defaults to 1 for all iterations in the search. The second is the way in which backtracking is performed. Backtracking is chronological in ACAS, best-first in ALS, and depth-first in DFBnB.

Table 7 shows the scheduling results of the STE algorithms for the 15 instances. While the average deviation across all instances grants the ALS as the best out of the three algorithms, the varying degrees of solution quality per instance class must be taken into account. From this perspective and benchmarking the performance as the problem size increases, the DFBnB worked better for larger instances. It outperformed the others in the medium and large instances obtaining the best solutions at a much reduced computation time in 7 out of 10 instances. Evidently, this makes it more practical than others. ACAS and ALSS only performed better in the small instances. Another reason DFBnB is the best fit for this problem is that the CPU time required to return the first solution is more or less stable (< 1 s) for all the instances, unlike the other two that experienced a sharp increase in time for the last three large instances. Also, all the first solutions obtained by DFBnB are clearly better.

STE algorithms are designed to produce feasible solutions quickly, and regularly improve the incumbent best solution C_{best} over time. They are able to guarantee optimality provided that the memory available and time allocated are large enough to reach the optimal solution. The incumbent best solution may have been obtained but cannot be considered optimal until all the markings with $f(M) \leq C_{best}$ have been expanded. As such, the time gap between when the incumbent best solution was returned and the convergence time varies depending on the number of markings remaining to be explored. For instance, DFBnB obtained the optimal solution for BGS3 at 1881.3 s but converged at 3213.9 s, while the gap is lower for ACAS and ALS that converged at 1370.3 s and 1364.7 s respectively for BGS4. On the other hand, both ACAS and ALS obtained the optimal solutions for BGS5 and BGM1 but did not converge within the CPU time limit.

4.4. Discussion

Each algorithm class has its strengths and weaknesses. The SE algorithms trade space for time. They are the best option when sufficient time is given for producing an optimal schedule. But it seems quite impractical for highly demanding and dynamic environments in which solutions must be returned in a short computation time. On the other hand, the STE algorithms offer an extra advantage in terms of both solution quality and time efficiency such that they can adapt to different memory and time constraints. One of the

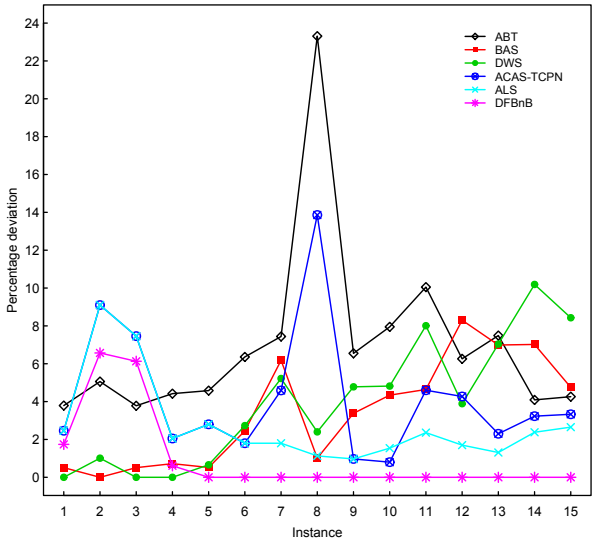


Figure 8: Relative percentage deviation of the first solution returned by the TE and STE classes.

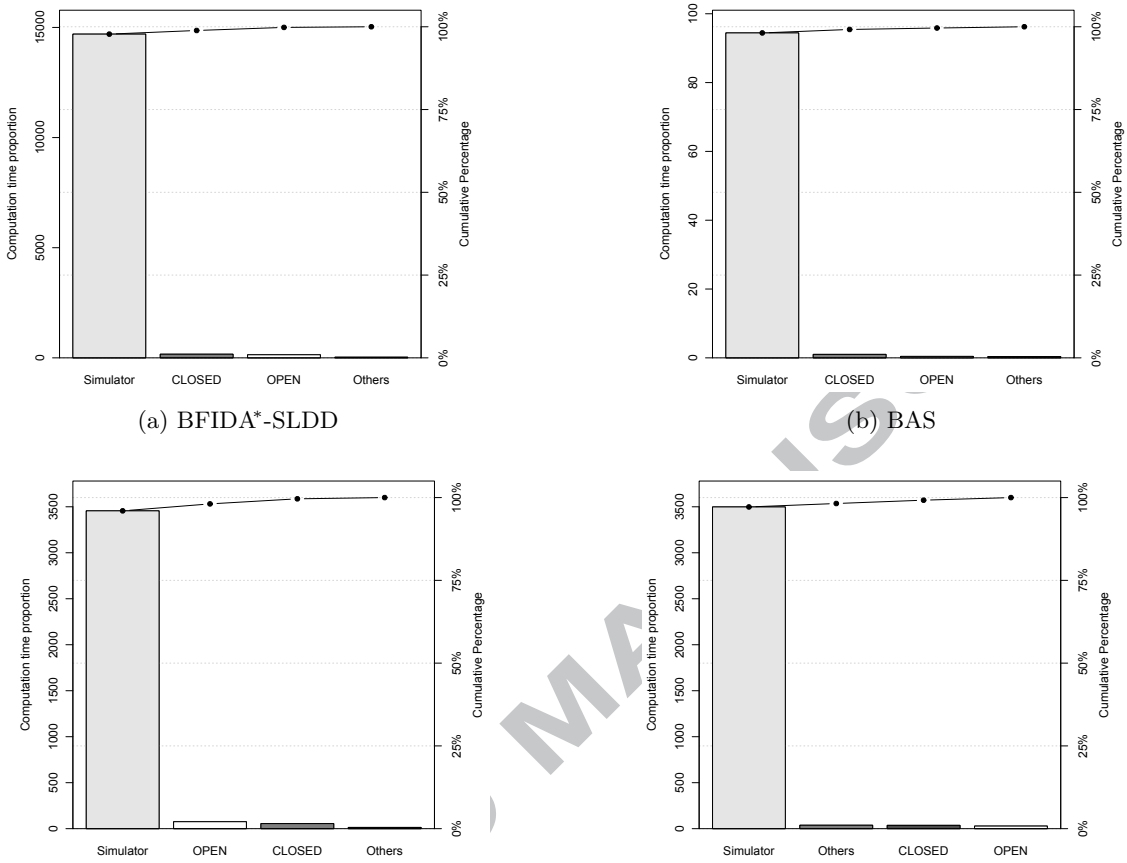
weaknesses of the STE class is that an additional running time does not necessarily lead to a better solution [73].

The SE class cannot be directly compared with TE (or vice-versa) because each stands at two ends of a continuum. However, the STE class can be benchmarked against SE in terms of the percentage of optimality lost (RPD from optimal solution) and the computation time reduction, and also against TE on solution quality and computation time comparison of the first solution obtained.

The optimality lost is quite low for the best performing algorithm in the STE class, DFBnB. It ranges between 0.3% and 2.1% for the non-converged solutions. The CPU time savings is about 99% for most of the instances excluding the first three. Figure 8 shows the RPD of the first solution returned by the TE and STE classes. The algorithm that produces the best first solution for each instance takes a zero RPD value. Clearly, the STE class outperforms the TE's with the exception of the small instances. The DFBnB performed better in nearly all the instances with the A*-BT signaled as the least performing.

For the TE and STE algorithms, it is quite difficult to predict when the first solution will be returned, as it largely depends on the problem size. Although DFBnB achieved a stable CPU time for all the instances. Notwithstanding, these results are not conclusive and cannot be used as a benchmark for all systems. The performance of each algorithm may be different for another problem set. Each system has its own behavior, and an empirical evaluation may be required to determine the best-performing algorithm. With these results, it is pretty straightforward to draw a conclusion on which algorithm can be adapted to an off-line or on-line scheduling when the system deviates from its original schedule or in the event of a failure or disturbance.

While the overall computation time depends on how each search algorithm explores the RG, it is worth benchmarking the time consumed on each computational task in the search exploration. To identify the main source of bottleneck in TIMSPAT, the distribution of the run time of four algorithms is given in Fig. 9. Clearly, the simulator dominates a larger proportion of the run time irrespective of the search algorithm employed. More time is spent on tasks like the enabling and firing of transitions for marking generation, and the computation of heuristic functions. This means that the overall efficiency of the tool relies on the simulator, which confirms that it is computationally expensive to simulate CPN models due to the difficulty in manipulating colors. The search part (OPEN and CLOSED) only consumes about 3% of the total time.



(a) BFIDA*-SLDD (b) BAS

(c) ALS (d) DFBnB

Figure 9: Run time proportion of computational tasks for the BGL1 instance.

5. Conclusion

In spite of the capability of RG to perform the automatic analysis of a modeled system, there is a lack of tool that supports the performance evaluation of TCPNs through timed state space analysis via HS methods. This motivated the development of TIMSPAT to deal with the shortcomings of existing tools. Most tools implementing the RG focus on the model checking of untimed nets. Other shortcomings include simulation limitations, timed state space generation with global clock synchronization, absence of efficient search algorithms, and reliance on third-party software applications.

The proposed tool provides a platform for describing CPN models as well as simulating the behavior of the system, and optimizing scheduling problems. One of the benefits presented by TIMSPAT is its ability to implement different heuristic search methods using the same syntax library and data structures. As a result, different scheduling scenarios can be benchmarked to allow for correct conclusions to be drawn. The tool is expected to support flexible decision making process without being over reliant on a particular solution algorithm. Several experiments have been performed on a real system with different algorithms, highlighting both their strengths and weaknesses. For real time scheduling purposes, the simulator can be easily integrated with the shop floor database to collect information on the current state of the system. It is worth mentioning that the tool is not limited to the already implemented algorithms.

Plans are already underway to incorporate metaheuristics such as genetic algorithms, ant colony optimization among others. Other future improvements to make the tool more robust for model development and execution includes:

- For a full-fledged simulator, data collection will be implemented in addition to appropriate selection of enabled transitions via random, priority or fair.
- A simulation-optimization framework to deal with uncertainties.
- The modeling approach allows the definition of new objective functions that can deal with scheduling policies oriented to lean manufacturing in which non-added-value operations would be minimized or a rapid manufacturing in which the total completion time would be minimized. This presents an opportunity to explore multiobjective optimization using the developed methodology.

References

[1] Aized, T. (2009). Modelling and performance maximization of an integrated automated guided vehicle system using coloured Petri net and response surface methods. *Computers & Industrial Engineering*, 57(3), 822–831, doi:10.1016/j.cie.2009.02.009.

[2] Aized, T. (2010). Modelling and analysis of multiple cluster tools system with random failures using coloured Petri net. *The International Journal of Advanced Manufacturing Technology*, 50(9-12), 897–906, doi:10.1007/s00170-010-2592-8.

[3] Amparore, E. (2014). A New GreatSPN GUI for GSPN Editing and CSLTA Model Checking. In G. Norman & W. Sanders (Eds.), *Quantitative Evaluation of Systems*, volume 8657 of *Lecture Notes in Computer Science* (pp. 170–173). Springer International Publishing.

[4] Baruwa, O. T. (2015). *A Timed State Space-Heuristic Search Framework for Colored Petri Net-based Scheduling of Discrete Event Systems – An Application to Flexible Manufacturing Systems*. PhD thesis, Autonomous University of Barcelona, Barcelona, Spain.

[5] Baruwa, O. T. (2016). TIMSPAT. `http://grupsderecerca.uab.cat/timspat/`. Accessed: 16-Mar-2016.

[6] Baruwa, O. T. & Piera, M. A. (2014). Anytime heuristic search for scheduling flexible manufacturing systems: a timed colored Petri net approach. *The International Journal of Advanced Manufacturing Technology*, 75(1-4), 123–137, doi:10.1007/s00170-014-6065-3.

[7] Baruwa, O. T. & Piera, M. A. (2015a). A coloured Petri net-based hybrid heuristic search approach to simultaneous scheduling of machines and automated guided vehicles. *International Journal of Production Research*, 0(0), 1–20, doi:10.1080/00207543.2015.1087656.

[8] Baruwa, O. T. & Piera, M. A. (2015b). Identifying FMS repetitive patterns for efficient search-based scheduling algorithm: A colored Petri net approach. *Journal of Manufacturing Systems*, 35(0), 120–135, doi:10.1016/j.jmsy.2014.11.009.

[9] Baruwa, O. T., Piera, M. A., & Guasch, A. (2015). Deadlock-Free Scheduling Method for Flexible Manufacturing Systems Based on Timed Colored Petri Nets and Anytime Heuristic Search. *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, 45(5), 831–846, doi:10.1109/TSMC.2014.2376471.

[10] Basak, O. & Albayrak, Y. E. (2015). Petri net based decision system modeling in real-time scheduling and control of flexible automotive manufacturing systems. *Computers & Industrial Engineering*, 86, 116126, doi:10.1016/j.cie.2014.09.024. Applications of Computational Intelligence and Fuzzy Logic to Manufacturing and Service Systems.

[11] Basile, F., Carbone, C., & Chiacchio, P. (2007). Simulation and analysis of discrete-event control systems based on Petri nets using PNetLab. *Control Engineering Practice*, 15(2), 241–259.

[12] Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., & Romijn, J. (2001). Efficient Guiding Towards Cost-Optimality in UPPAAL. In T. Margaria & W. Yi (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science* (pp. 174–188). Springer Berlin Heidelberg.

[13] Behrmann, G., Larsen, K. G., & Rasmussen, J. I. (2005). Optimal Scheduling Using Priced Timed Automata. *SIGMETRICS Perform. Eval. Rev.*, 32(4), 34–40, doi:10.1145/1059816.1059823.

[14] Bensalem, S., Havelund, K., & Orlandini, A. (2014). Verification and validation meet planning and scheduling. *International Journal on Software Tools for Technology Transfer*, 16(1), 1–12, doi:10.1007/s10009-013-0294-x.

[15] Bodenstein, C. & Zimmermann, A. (2014). TimeNET Optimization Environment: Batch Simulation and Heuristic Optimization of SCPNs with TimeNET 4.2. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '14 (pp. 129–133). ICST, Brussels, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[16] Christensen, S., Kristensen, L. M., & Mailund, T. (2001). Condensed State Spaces for Timed Petri Nets. In J.-M. Colom & M. Koutny (Eds.), *Applications and Theory of Petri Nets 2001*, volume 2075 of *Lecture Notes in Computer Science* (pp. 101–120). Springer Berlin Heidelberg.

[17] Cimatti, A., Edelkamp, S., Fox, M., Magazzeni, D., & Plaku, E. (2015). Automated Planning and Model Checking. *Dagstuhl Reports*, 4(11), 227–245.

[18] Davidrajuh, R. (2015). Benchmarking GPenSIM. In K. Elleithy & T. Sobh (Eds.), *New Trends in Networking, Computing, E-learning, Systems Sciences, and Engineering*, volume 312 of *Lecture Notes in Electrical Engineering* (pp. 373–379). Springer International Publishing.

[19] Davidrajuh, R. & Lin, B. (2011). Exploring airport traffic capability using Petri net based model. *Expert Systems with Applications*, 38(9), 10923–10931.

[20] Denaro, G. & Pezzè, M. (2004). *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, chapter Petri Nets and Software Engineering, (pp. 439–466). Springer Berlin Heidelberg: Berlin, Heidelberg.

[21] Dingle, N. J., Knottenbelt, W. J., & Suto, T. (2009). PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets. *SIGMETRICS Perform. Eval. Rev.*, 36(4), 34–39, doi:10.1145/1530873.1530881.

[22] Edelkamp, S. & Jabbar, S. (2006). Action Planning for Directed Model Checking of Petri Nets. *Electronic Notes in Theoretical Computer Science*, 149(2), 3 – 18, doi:10.1016/j.entcs.2005.07.023. Proceedings of the Third Workshop on Model Checking and Artificial Intelligence (MoChArt 2005)Model Checking and Artificial Intelligence 2005.

[23] Evangelista, S. & Pradat-Peyre, J.-F. (2004). An efficient algorithm for the enabling test of colored Petri nets. In *Fifth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, number 570 (pp. 137–156).

[24] Gaeta, R. (1996). Efficient discrete-event simulation of colored Petri nets. *Software Engineering, IEEE Transactions on*, 22(9), 629–639, doi:10.1109/32.541434.

[25] Gallasch, G. E. & Billington, J. (2009). Relaxed Timed Coloured Petri Nets – A Motivational Case Study. In *Tenth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, volume 590 (pp. 209–218). Aarhus, Denmark.

[26] Ghosh, K., Dasgupta, P., & Ramesh, S. (2015). Automated Planning as an Early Verification Tool for Distributed Control. *Journal of Automated Reasoning*, 54(1), 31–68, doi:10.1007/s10817-014-9313-1.

[27] Harjunkoski, I., et al. (2014). Scope for industrial applications of production scheduling models and solution methods. *Computers & Chemical Engineering*, 62, 161 – 193, doi:10.1016/j.compchemeng.2013.12.001.

[28] He, X. & Wu, Z. (2011). Deadlock-free assignment of wafer processing in photolithography equipment - by using a CPN model. *Transactions of the Institute of Measurement and Control*, 33(3-4), 422–434, doi:10.1177/0142331208100098.

[29] Huang, B., Jiang, R., & Zhang, G. (2014). Search strategy for scheduling flexible manufacturing systems simultaneously using admissible heuristic functions and nonadmissible heuristic functions. *Computers & Industrial Engineering*, 71(0), 21–26, doi:10.1016/j.cie.2014.02.010.

[30] Huang, B., Sun, Y., & Sun, Y. M. (2008). Scheduling of flexible manufacturing systems based on Petri nets and hybrid heuristic search. *International Journal of Production Research*, 46(16), 4553–4565.

[31] Huang, B., Sun, Y., Sun, Y.-M., & Zhao, C.-X. (2010). A hybrid heuristic search algorithm for scheduling FMS based on Petri net model. *The International Journal of Advanced Manufacturing Technology*, 48(9-12), 925–933.

[32] Jensen, K., Kristensen, L., & Wells, L. (2007). Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9, 213–254, doi:10.1007/s10009-007-0038-x.

[33] Jensen, K. & Kristensen, L. M. (2009). *Coloured Petri nets: modelling and validation of concurrent systems*. Springer.

[34] Julvez, J., Matcovschi, M., & Pastravanu, O. (2014). MATLAB tools for the analysis of Petri net models. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE* (pp. 1–12).

[35] Kordon, F., et al. (2015). Complete Results for the 2015 Edition of the Model Checking Contest. `http://mcc.lip6.fr/2015/results.php`.

[36] Kordon, F., et al. (2012). Report on the Model Checking Contest at Petri Nets 2011. *Transactions on Petri Nets and Other Models of Concurrency*.

[37] Kounev, S. & Buchmann, A. (2006). SimQPN-A tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation*, 63(4-5), 364–394.

[38] Kristensen, L., Mechlenborg, P., Zhang, L., Mitchell, B., & Gallasch, G. (2008). Model-based development of a course of action scheduling tool. *International Journal on Software Tools for Technology Transfer*, 10(1), 5–14, doi:10.1007/s10009-007-0053-y.

[39] Lakos, C. & Petrucci, L. (2007). Modular state space exploration for timed Petri nets. *International Journal on Software Tools for Technology Transfer*, 9(3-4), 393–411.

[40] Latorre-Biel, J., Jiménez-Macías, E., Blanco-Fernández, J., Martínez-Cámara, E., Sáenz-Díez, J., & Pérez-Parte, M. (2015). Decision Support System, Based on the Paradigm of the Petri Nets, for the Design and Operation of a Dairy Plant. *International Journal of Food Engineering*, 11(6), 767–776.

[41] Lee, J. & Lee, J. S. (2010). Heuristic search for scheduling flexible manufacturing systems using lower bound reachability matrix. *Computers & Industrial Engineering*, 59(4), 799–806.

[42] Lei, H., Xing, K., Han, L., Xiong, F., & Ge, Z. (2014). Deadlock-free scheduling for flexible manufacturing systems using Petri nets and heuristic search. *Computers & Industrial Engineering*, 72(0), 297–305.

[43] Li, C., Wu, W., Feng, Y., & Rong, G. (2015). Scheduling FMS problems with heuristic search function and transition-timed Petri nets. *Journal of Intelligent Manufacturing*, 26(5), 933–944, doi:10.1007/s10845-014-0943-2.

[44] Li, Y., Dong, J. S., Sun, J., Liu, Y., & Sun, J. (2014). Model checking approach to automated planning. *Formal Methods in System Design*, 44(2), 176–202, doi:10.1007/s10703-013-0197-1.

[45] Liu, F. & Heiner, M. (2010). Computation of enabled transition instances for colored Petri nets. In *17th German Workshop on Algorithms and Tools for Petri Nets (AWPN)*, volume 643 (pp. 51–65).

[46] Makela, M. (2000). Modular Reachability Analyzer for High-Level Petri Nets. In R. Boel & G. Stremersch (Eds.), *Discrete Event Systems*, volume 569 of *The Springer International Series in Engineering and Computer Science* (pp. 477–478). Springer US.

[47] Malone, B. & Yuan, C. (2014). A Depth-First Branch and Bound Algorithm for Learning Optimal Bayesian Networks. In M. Croitoru, S. Rudolph, S. Woltran, & C. Gonzales (Eds.), *Graph Structures for Knowledge Representation and Reasoning*, volume 8323 of *Lecture Notes in Computer Science* (pp. 111–122). Springer International Publishing.

[48] Mejia, G., Nino, K., Montoya, C., Sanchez, M. A., Palacios, J., & Amodeo, L. (2016). A Petri Net-based framework for realistic project management and scheduling: An application in animation and videogames. *Computers & Operations Research*, 66, 190 – 198, doi:10.1016/j.cor.2015.08.011.

[49] Mejia, G. & Odrey, N. G. (2005). An approach using Petri nets and improved heuristic search for manufacturing system scheduling. *Journal of Manufacturing Systems*, 24(2), 79–92.

[50] Moro, A., Yu, H., & Kelleher, G. (2000). Advanced scheduling methodologies for flexible manufacturing systems using Petri nets and heuristic search. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3 (pp. 2398–2403 vol.3).

[51] Mujica, M. A. & Piera, M. A. (2010). Performance optimisation of a CNC machine through exploration of timed state space. *International Journal of Simulation and Process Modelling*, 6(2), 165–174.

[52] Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4), 541–580.

[53] Narciso, M. E., Piera, M. A., & Guasch, A. (2012). A time stamp reduction method for state space exploration using colored Petri nets. *Simulation*, 88(5), 592–616.

[54] Nishi, T. & Wakatake, M. (2014). Decomposition of timed automata for solving scheduling problems. *International Journal of Systems Science*, 45(3), 472–486, doi:10.1080/00207721.2012.724099.

[55] Panek, S., Stursberg, O., & Engell, S. (2006). Efficient synthesis of production schedules by optimization of timed automata. *Control Engineering Practice*, 14(10), 1183 – 1197, doi:10.1016/j.conengprac.2006.02.014. The Seventh Workshop On Discrete Event Systems (WODES2004)The Seventh Workshop On Discrete Event Systems (WODES2004).

[56] Petri nets world (2015). Petri nets tool database. `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html`. Accessed: 14-Feb-2015.

[57] Piera, M. & Music, G. (2011). Coloured Petri net scheduling models: Timed state space exploration shortages. *Mathematics and Computers in Simulation*, 82(3), 428–441.

[58] PNML, S. (2016). Petri Net Markup Language website. `http://www.pnml.org/`. Accessed: 14-May-2016.

[59] Reyes, A., Yu, H., Kelleher, G., & Lloyd, S. (2002). Integrating Petri Nets and Hybrid Heuristic Search for the Scheduling of FMS. *Comput. Ind.*, 47(1), 123–138.

[60] Russell, S. J. & Norvig, P. (2009). *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall.

[61] Ruys, T. C. (2003). *Model Checking Software: 10th International SPIN Workshop Portland, OR, USA, May 9–10, 2003 Proceedings*, chapter Optimal Scheduling Using Branch and Bound with SPIN 4.0, (pp. 1–17). Springer Berlin Heidelberg: Berlin, Heidelberg.

[62] Schoppmeyer, C., Subbiah, S., Valdes, J. M. D. L. F., & Engell, S. (2014). Dynamic Scheduling of Shuttle Robots in the Warehouse of a Polymer Plant Based on Dynamically Configured Timed Automata Models. *Industrial & Engineering Chemistry Research*, 53(44), 17135–17154, doi:10.1021/ie500437r.

[63] Tuncel, G. & Bayhan, G. (2007). Applications of Petri nets in production scheduling: a review. *The International Journal of Advanced Manufacturing Technology*, 34(7-8), 762–773.

[64] van der Aalst, W. (1993). Interval timed coloured Petri nets and their analysis. In M. Ajmone Marsan (Ed.), *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science* (pp. 453–472). Springer Berlin Heidelberg.

[65] Vempaty, N. R., Kumar, V., & Korf, R. E. (1991). Depth-First Versus Best-First Search. In *AAAI* (pp. 434–440).

[66] Viswanadham, N. & Narahari, Y. (1987). Coloured Petri net models for automated manufacturing systems. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4 (pp. 1985–1990).

[67] Wang, P., Ma, L., Goverde, R., & Wang, Q. (2016). Rescheduling Trains Using Petri Nets and Heuristic Search. *Intelligent Transportation Systems, IEEE Transactions on*, 17(3), 726–735, doi:10.1109/TITS.2015.2481091.

[68] Westergaard, M., Evangelista, S., & Kristensen, L. (2009). ASAP: An Extensible Platform for State Space Analysis. In G. Franceschinis & K. Wolf (Eds.), *Applications and Theory of Petri Nets*, volume 5606 of *Lecture Notes in Computer Science* (pp. 303–312). Springer Berlin Heidelberg.

[69] Westergaard, M. & Verbeek, H. E. (2011). Efficient Implementation of Prioritized Transitions for High-level Petri Nets. In *International Workshop on Petri Nets and Software Engineering*, volume 723 (pp. 27–41). Newcastle upon Tyne, UK.

[70] Xie, C. & Allen, T. T. (2015). Simulation and experimental design methods for job shop scheduling with material handling: a survey. *The International Journal of Advanced Manufacturing Technology*, (pp. 1–11)., doi:10.1007/s00170-015-6981-x.

[71] Xiong, H. H. & Zhou, M. (1998). Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search. *Semiconductor Manufacturing, IEEE Transactions on*, 11(3), 384–393.

[72] Yu, H., Reyes, A., Cang, S., & Lloyd, S. (2003). Combined Petri net modelling and AI-based heuristic hybrid search for flexible manufacturing systems-part II. Heuristic hybrid search. *Computers & Industrial Engineering*, 44(4), 545–566.

[73] Zamani, R. (2010). A parallel complete anytime procedure for project scheduling under multiple resource constraints. *The International Journal of Advanced Manufacturing Technology*, 50(1-4), 353–362.

[74] Zhang, W. (1999). Truncated and anytime depth-first branch-and-bound: A case study on the asymmetric Traveling Salesman Problem. In *AAAI 1999 Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information* (pp. 148–153). Stanford, CA.

[75] Zhang, W. & Korf, R. E. (1995). Performance of linear-space search algorithms. *Artificial Intelligence*, 79(2), 241–292, doi:10.1016/0004-3702(94)00047-6.

[76] Zhou, R. & Hansen, E. A. (2006). Breadth-first heuristic search. *Artificial Intelligence*, 170(4-5), 385–408.

[77] Zimmermann, A. (2012). Modeling and evaluation of stochastic Petri nets with TimeNET 4.1. In *Performance Evaluation Methodologies and Tools (VALUETOOLS), 2012 6th International Conference on* (pp. 54–63).

[78] Zuberek, W. M. (1996). *Modeling using timed Petri nets - discrete-event simulation*. Technical report, Memorial University of Newfoundland, Department of Computer Science, Memorial University, St. John's, Canada A1B 3X5.

[79] Zuberek, W. M. (2015). TPN-Tools. `http://www.cs.mun.ca/~wlodek/research/proj-tpn-tools.html`. Accessed: 15-Dec-2015.

[80] Zurawski, R. & Zhou, M. (1994). Petri nets and industrial applications: A tutorial. *Industrial Electronics, IEEE Transactions on*, 41(6), 567–583.

- We present a reachability graph-based search optimization tool for scheduling.
- Motivated by the lack of tool support for optimization of TCPNs.
- Implements an event-driven timed state space with AI heuristic search algorithms.
- Aimed at supporting flexible decision making process with algorithm portfolio.
- Comparative study of nine search algorithms on real system demonstrates tool efficiency.