Deadlock-Free Scheduling Method for Flexible Manufacturing Systems Based on Timed Colored Petri Nets and Anytime Heuristic Search

Olatunde T. Baruwa, Miquel Angel Piera, and Antoni Guasch

***Abstract*—This paper addresses the deadlock (DL)-free scheduling problem of flexible manufacturing systems (FMS) characterized by resource sharing, limited buffer capacity, routing flexibility, and the availability of material handling systems. The FMS scheduling problem is formulated using timed colored Petri net (TCPN) modeling where each operation has a certain number of preconditions, an estimated duration, and a set of postconditions. Based on the reachability analysis of TCPN modeling, we propose a new anytime heuristic search algorithm which finds optimal or near-optimal DL-free schedules with respect to makespan as the performance criterion. The methodology tackles the time-constrained problem of very demanding systems (high diversity production and make-to-order) in which computational time is a critical factor to produce optimal schedules that are DL-free. In such a rapidly changing environment and under tight customer due-dates, producing optimal schedules becomes intractable given the time limitations and the NP-hard nature of scheduling problems. The proposed anytime search algorithm combines breadth-first iterative deepening A* with suboptimal breadth-first heuristic search and backtracking. It guarantees that the search produces the best solution obtained so far within the allotted computation time and provides better solutions when given more time. The effectiveness of the approach is evaluated on a comprehensive benchmark set of DL-prone FMS examples and the computational results show the superiority of the proposed approach over the previous works.**

***Index Terms*—Anytime heuristic search, deadlock-free, flexible manufacturing systems (FMS), Petri nets (PN), reachability analysis, scheduling.**

I. Introduction

FLEXIBLE manufacturing systems (FMS) can efficiently produce several part types simultaneously using various automated transport and production resources such as robots and computer numerically controlled machines. Products are manufactured according to an ordered (defined) sequence of operations or tasks. The operation of some jobs may be processed by more than one machine (routing flexibility) or certain machines may be used to process more than one job type (machine flexibility). When an operation is completed, parts may be moved to the next machine through material handling systems (MHS) (conveyors, robots, pallets, and automated guided vehicles) or transferred to intermediate buffer facilities. The increased flexibility offered by an FMS can provide greater productivity [1] and allows the system to adapt rapidly to changes in production plans in response to changing demand patterns. However, this gives rise to a scheduling problem, which involves the efficient allocation of resources over time to perform a collection of tasks (jobs) considering capacity and precedence constraints where one or several objectives have to be optimized [2]. For example, the minimization of the makespan or total completion time, the manufacturing lead time, or the work in process are some of the most well-accepted objectives.

The scheduling algorithms developed for traditional manufacturing systems (job shop, flow shop, etc.) assume infinite buffer capacity and do not take into account material handling systems [3], [4]. In practice, however, the allocation of resources in an FMS is usually difficult to manage due to inherent resource limitations like no buffer storage, limited buffer capacity, and MHS availability. The concurrent competition for a finite number of resources in a manufacturing environment usually results in deadlock (DL) situations, which bring the system to a permanent halt state. A DL occurs when a set of parts are in "circular waiting," i.e., each part in the set waits for a resource held by another part in the same set [5]. DL problems in FMS cause poor resource utilization, production inefficiencies, long down-time, etc., and can drastically affect other key performance indicators. Therefore, it is crucial to develop appropriate resource allocation and scheduling techniques to optimize the system performance while preventing DL situations [6], [7].

DL resolution strategies can be seen from either control or scheduling perspective. Most of the research works on DL resolution have focused more on control strategies vis-à-vis prevention [8]–[16], avoidance [5], [17]–[19], and

detection and recovery [20], [21]. Li *et al.* [22], [23] provide a comprehensive state-of-the-art review of DL control methods for FMS. However, DL-free scheduling has received little attention in the literature [7]. Control methods try to avoid or prevent the system from entering DL situations in order to guarantee DL-free operations. Unfortunately, these methods are not suitable to optimize system performance since they do not take operation times into account [7], [24].

DL-free scheduling problems of FMS have been formulated using three modeling approaches: 1) mathematical programming [25]; 2) disjunctive graph [26]; and 3) discrete-event system modeling [3], [6], [7], [24], [27]–[32]. Ramaswamy and Joshi [25] present mathematical formulations for the DL-free scheduling of an automated manufacturing workstation that considers material handling and finite buffer capacity. Because of solution time considerations, a Lagrangian relaxation heuristic is used to extend the basic formulations to solve more realistic problems. However, practical constraints related to MHS, resource sharing situations, DLs, finite buffer sizes, multiple lot sizes, and routing flexibility are usually difficult to describe mathematically [24], [33]. Mati *et al.* [26] extend disjunctive graph representation to job shop scheduling with blocking and use a taboo search based on neighborhood structure to find optimal DL-free schedules.

In the discrete-event system approach, two modeling formalisms are commonly used because of their capability to explicitly represent DL states in an FMS: 1) Petri net (PN) [3], [24], [34], [35] and 2) automata (AA) [27], [36], [37]. PNs have been used extensively to model, simulate, and analyze manufacturing systems where there is a high level of concurrency, parallelism, causal dependency, and synchronization [33]. In this paper, FMS are formalized using timed colored PN (TCPN) modeling (a high-level PN with time extension) in which the structure, processes, and functionality for each product in a high diversity production environment can be understood from a discrete-event system approach. In such an approach, each operation has a certain number of preconditions, an estimated duration, and a set of postconditions. Colored PN (CPN) modeling is preferred since it provides a more compact representation of the system while maintaining the same modeling power of PN.

Two different approaches have been developed to deal with DL-free scheduling based on PN modeling. The first approach [3], [24], [28], [29], [38], [39] performs scheduling on DL-prone PNs and only takes into account DL situations during the scheduling process. DL detection phases are incorporated into their optimization procedures. Here, the scheduling problem is formulated as a search problem through the generation of the reachability graph (or the state space) of a DL-prone PN. The objective is to find the optimal sequence of transition firings from the initial marking to a defined goal marking which minimizes a given performance measure. A DL-free schedule is guaranteed if the firing sequence reaches the defined goal marking. As the graph is generated, DL markings are detected if no transition is enabled. The paths leading to these markings are marked and they are not considered for further exploration. Reachability graph analysis is a reliable and efficient method to obtain optimal schedules; however, the computational complexity is so high that it is practically impossible to explore the full state space of an industrial-sized system due to the explosion problem. Artificial intelligence (AI)-based heuristic search methods [3], [24], [28], [29], [38], [39] have been proposed to simulate only the best scenarios by generating partial reachability graphs with heuristic functions.

The second approach [6], [7], [31], [32], [40], [41] introduces DL control policies [8]–[11], [19] into DL-prone PNs to make them live (or DL-free) before scheduling is carried out. A live PN ensures that no DL state is reached during the system evolution. Based on the live PN, an optimal DL-free schedule is obtained using search algorithms. Structural PN objects, namely siphons [8]–[11], [42], [43] and resource-transition circuits [19], [42], have been used to characterize DLs and develop DL control policies for FMS. For example, siphons are implemented by adding control places and arcs to enforce liveness in the PN such that no siphon can be emptied (a necessary and sufficient condition for a DL-free PN).

However, this integration requires that DL control policies must be optimal, i.e., they must consider all the possible rules or scenarios that do not restrict the firing of a feasible transition that can lead to a better scheduling performance. Since the computation procedure of optimal control policies is NP-hard [7], [31], [42], these methods [7], [31] are system-specific and more suitable to a particular set of FMS problems where the capacity of the same kind of resource and lot sizes of parts are larger. References [8]–[11] and [42] have focused on how to efficiently compute siphons and design a live PN. Also, the controllers are conservative and impose constraints on the system by considering a limited number of alternatives, which in turn affects the system's performance. Genetic search algorithms [7], [31], [32] and reachability graph methods [6], [44] have been used to search for near-optimal DL-free schedules. Another approach [27] based on AA uses Ramadge–Wonham's [45] supervisory control theory to synthesize a DL-free controller for FMS models, and A* heuristic search for scheduling. This paper focuses on the first approach using CPN since the control rules in PN cannot be easily obtained in CPN.

Besides the state explosion problem, one of the challenges of the reachability graph is how to deal with a large number of DLs that can occur during generation. This prevents solutions from being returned at reasonable computation time and may saturate the memory very quickly. Also, it can take a considerable amount of time to obtain optimal schedules with classical heuristic search algorithms like A* even when the memory required is sufficient to explore the full state space. Several heuristic search methods [3], [24], [28], [39] do not return a solution until termination and they may fail to return one at memory run out for sizable problems. In addition, the run-time overhead for PN-based methods using reachability graph is usually quite high since the time spent to obtain an optimal DL-free schedule includes not only the generation of the state space but also DL detection and node evaluation [27].

Owing to the time-critical decision making and deadline pressure in production scheduling environments, a limited amount of time is given to produce optimal or near-optimal DL-free schedules. Since scheduling is an operational decision-making activity that must be dealt with in a short-term horizon, computation time becomes a critical factor. In such a rapidly changing environment and under tight customer due-dates, producing optimal solutions is considered intractable given the time limitations and the NP-hard nature of scheduling problems. Thus, it is more appropriate to accept suboptimal solutions if the computation time can be reduced.

Anytime heuristic search algorithms have been proposed in the AI domain to work under time-critical conditions [46]–[49]. Rather than waiting for the search to terminate, a first suboptimal solution is found quickly. This solution is then improved continuously over time until the search converges to an optimal solution, provided the computation time is sufficient to prove optimality. Basically, they trade off computation time and solution quality (SQ) to overcome the drawbacks of classical heuristic search algorithms that often take a long time to find optimal solutions to large-sized problems. Anytime algorithms guarantee that the search produces the best solution obtained so far within the allotted computation time and they are expected to return better solutions when given more time. A recent study [50] adapted the anytime column search [51] to schedule FMS without considering DL situations. The proposed approach called anytime layered search (ALS) combines breadth-first iterative deepening A* search (BFIDA*) with suboptimal breadth-first heuristic search (sBFHS) [52] and backtracking to provide a sequence of improving solutions. The underlying search space of the ALS is based on the condensed state space (CSS) of TCPN models, which provides a compact representation of the timed state space using untimed marking equivalence.

The rest of this paper is organized as follows. Section II gives the background to TCPN modeling using a practical example of DL-prone FMS and an illustrative scenario on how to obtain a DL-free model with control policies. Section III introduces the scheduling of FMS based on reachability graph analysis and CSS of TCPN models. Section IV describes the proposed ALS algorithm for DL-free scheduling while Section V evaluates and compares the performance of the ALS method on an extensive set of DL-prone examples with state-of-the-art algorithms. Section VI concludes this paper by giving a summary of the research and ideas for future work.

II. FMS Modeling With TCPN

A CPN is a directed bipartite graph with two node types called places and transitions where the nodes are connected via directed arcs. In a CPN, the advantages of classical PNs are enhanced by a condensed representation of the model with the use of a data value called colored token. In an FMS description, a place containing tokens represents a job or resource status, a transition describes the event (the start or completion) that may occur (or fire) based on the preconditions of input arc expressions and guards, while an arc models the flow of parts. Graphically, places, transitions, arcs, and guards are represented by circles, boxes, arrows, and square brackets, respectively.

To conduct performance analysis, a CPN is extended with a time concept based on the introduction of a global clock. The global clock represents the model time and each token has a time attribute called the time stamp. The time stamp describes the earliest time at which a token becomes available (ready to be used). The time representation can be used to model activity durations, earliness, delays, deadlines, etc. Deterministic time values are considered in this paper.

Definition 1 [53]: Formally, a timed nonhierarchical CPN can be defined as a nine-tuple, TCPN $= (P, T, A, \sum, V, C, G, E, I)$ where:

P	finite set of places $\{p_1, p_2, \ldots, p_m\}$;
T	finite set of transitions $\{t_1, t_2, \ldots, t_n\}$ such that $P \cap T = \emptyset$;
A	finite set of directed arcs $\{a_1, a_2, \ldots, a_k\}$ such that $A \subseteq P \times T \cup T \times P$;
$\sum$	finite set of nonempty types called colored sets that determine the data values (tokens) and the operations and functions that can be used in the net inscriptions (i.e., arc, guard, and initialization expressions). Each color set is either timed or untimed;
V	finite set of typed variables such that $\text{Type}[v] \in \sum$ for all variables $v \in V$;
$C : P \to \sum$	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;
$G : T \to \text{EXPR}_v$	guard function that assigns a guard to each transition t such that $\text{Type}[G(t)] = \text{Bool}$;
$E : A \to \text{EXPR}_v$	arc expression function that assigns an arc expression to each arc a such that $\text{Type}[E(a)] = C(p)_{\text{MS}}$ if p is untimed and $\text{Type}[E(a)] = C(p)_{TMS}$ if p is timed, where p is the place connected to the arc and C_{MS} denotes the set of all multisets over C;
$I : P \to \text{EXPR}_\emptyset$	initialization function that assigns an initialization expression to each place p such that $\text{Type}[I(p)] = C(p)_{\text{MS}}$ if p is untimed and $\text{Type}[I(p)] = C(p)_{TMS}$ if p is timed.

The current state of the system is defined by the distribution of tokens over the places called marking. A marking maps each place into a timed multiset of token elements and a timed marking is a pair (M, r^*), which consists of the marking M together with the time stamps of the tokens and $r^* \in \mathbb{R}$ the value of the global clock [53]. The initial timed marking M_0 consists of the markings of each place (written above the place) in the model representing the initial state of the system. In a TCPN, transitions are associated with a delay interpreted as machine processing or transportation time in a manufacturing environment, represented as "@+time value." Also, the TCPN formalism allows time delays to be specified for places

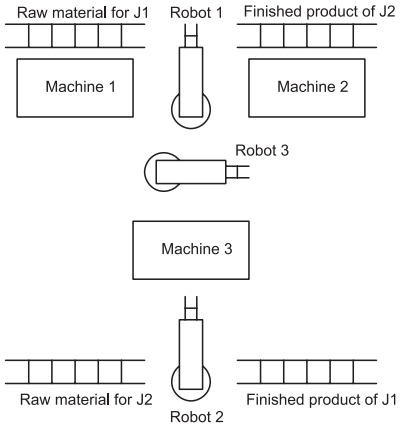

Fig. 1. Layout of FMS Example 1 [27].

TABLE I
ROUTING AND PROCESSING TIMES OF JOBS

Job/Operation	1	2	3
J_1	$M_1(5)$	$M_2(8)$	$M_3(2)$
J_2	$M_3(7)$	$M_1(3)$	$M_2(9)$

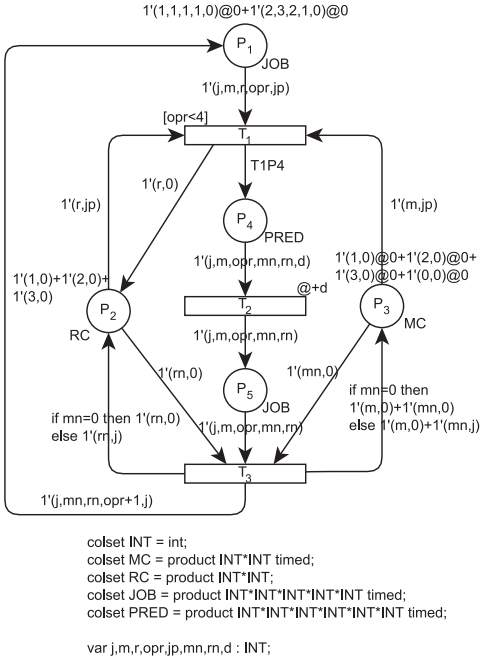

Fig. 2. TCPN model of FMS Example 1.

or arcs. This paper focuses on only transition delays. The interested reader is referred to [53] for a detailed tutorial on CPN formulations and theory.

Example 1: To illustrate the TCPN modeling of a DL-prone system, let us consider an FMS example [28] of a no-buffer system (with blocking) where each job has to wait on a machine until it can be processed on the next machine. The loading, unloading, and transportation operations are performed with multiple shared MHS (robots). The FMS consists of three machines M_1, M_2, and M_3 and three robots R_1, R_2, and R_3 with two job types J_1 and J_2 to be produced. The layout of the FMS is shown in Fig. 1 and the routing and processing times of jobs are given in Table I. In Table I, $M_1(5)$ means the first operation of job J_1 must be performed on machine M_1 during five time units. Each machine can process only one part at a time. The robots move parts between machines and each one can transport only one part at a time. Robot R_1 is shared by M_1 and M_2 and can be used to load raw material of product J_1 to M_1, and unload finished product J_2 from M_2. Robot R_2 is used to load M_3, to deliver raw material of product J_2, and unload finished product J_1 from M_3. Robot R_3 is shared by the three machines to transport intermediate parts. The loading and unloading times are not considered as a decision variable.

Fig. 2 shows the TCPN model of the FMS example together with the color sets and variables. The TCPN consists of five places ($P_1, P_2, \ldots, P_5$) and three transitions (T_1, T_2, T_3). The interpretation of the places (including the conditional arc expression $T1P4$) and colors is given in Tables II and III, respectively. In the model, the symbol *&* is used as the equivalent of *andalso* and a single $+$ for the multiset of tokens instead of the standard $++$. The arc expression $T1P4$ describes the routing and processing times of jobs. It is not placed in the figure for legibility. The cardinality (weight of each token) in place P_1 represents the lot size of the corresponding job.

TABLE II
PLACES AND THEIR MEANINGS

Place	Meaning
P_1	Raw materials available at the loading station for operation 1 or ready to be loaded or currently being transported by robot to the machine for the next operation
P_2	Robot idle or busy
P_3	Machine idle or busy
P_4	Job ready to be processed by machine
P_5	Jobs waiting to be transported by robot to machine for the next operation or finished products waiting to be unloaded
$T1P4$	if $j = 1$ & $opr = 1$ then $1'(j, m, opr, 2, 3, 5)$ else if $j = 1$ & $opr = 2$ then $1'(j, m, opr, 3, 3, 8)$ else if $j = 1$ & $opr = 3$ then $1'(j, m, opr, 0, 2, 2)$ else if $j = 2$ & $opr = 1$ then $1'(j, m, opr, 1, 3, 7)$ else if $j = 2$ & $opr = 2$ then $1'(j, m, opr, 2, 3, 3)$ else $1'(j, m, opr, 0, 1, 9)$

The same model can be used for lot sizes of job greater than 1 by changing the weight of each token in place P_1. Place P_2 is the only untimed place in the net since the transportation times of robots are ignored. The loading (unloading) of the part onto (from) the machine before (after) the operation is regarded as one operation [1]. There are 2, 3, and 4 tokens in places P_1, P_2, and P_3 respectively, while the others are empty. Token $1'(0, 0)$ is used as a dummy machine to synchronize the transportation of the job to the unloading station after the last operation is performed. As seen in the output arc expression $T3P1$, the value of the operation type color *opr* is increased when the part is being unloaded from the machine for the next operation.

The time stamps of the tokens are written after the symbol @ and the global clock is at time 0. The time stamps of the tokens in places P_1, P_4, and P_5 represent the time at which a job starts or finishes its last operation while those in place P_3 represent the time at which a machine finishes processing a job or when the last job was unloaded from the machine. The untimed marking M_u of M_0, i.e., $M_0(M_u)$ is obtained by

TABLE III
COLORS AND THEIR MEANINGS

Color	Meaning
j	Job type
opr	Operation type identifier
m	Machine identifier
r	Robot identifier
jp	Job type currently being loaded at machine or transported by robot (0 - raw material or machine/robot is available)
mn	Machine identifier to process job for next operation
rn	Robot identifier to transport job for next operation
d	Processing time of operation

removing all the time stamps from the tokens in places P_1, P_3, P_4, and P_5.

Transition T_1 describes the loading process event when a job is being transported from the load station or from a machine after processing. Transition T_2 represents the execution of a job's task in a machine with an assigned duration of $@+d$, the start and completion of the job processing, while T_3 describes the unloading process of jobs from the machines. T_1 and T_3 are immediate transitions without duration as transportation time is omitted. They are implicitly assigned a zero time delay $@+0$.

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 [54]. For example, if transition T_1 is enabled by the tokens $1'(1,1,1,1,0)@0$, $1'(1,0)$, and $1'(1,0)@12$ in places P_1, P_2, and P_3, respectively, T_1 is said to be enabled at time 12. 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. This means that the tokens are removed from the input places and added to the output places of the firing transition. The number and color of the tokens are determined by the arc expressions, evaluated for the occurring bindings [53]. 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 $\tau_k + d$.

For transition T_1 to be enabled, the machine assigned to process a job for the next operation must be free or be tied to a specific job during the unloading process of the same job from the previous operation, the assigned robot is available or currently transporting the job from the previous operation and the number of operations required to complete the job is less than 4 ($opr < 4$). Transition T_3 is enabled if both the assigned machine and the robot to transport the job for the next operation are free. Otherwise, the job continually waits at the machine used for the current operation until the resources become available. Also, this impedes another operation to be performed at this machine if another job needs to be processed on it. This situation can lead to a DL. A marking M is called a DL marking if no transition is enabled in M. For example, if the TCPN model in Fig. 2 is simulated with a lot size of 2 for each job and it reaches a marking M_{12}

P_1 $1'(1,1,1,1,0)@0$; P_2 $1'(1,0)++1'(2,0)++1'(3,0)$; P_3 $1'(0,0)@0$

P_4 empty; P_5 $1'(1,2,2,3,3)@13++1'(2,1,2,2,3)@10++1'(2,3,1,1,3)@14$

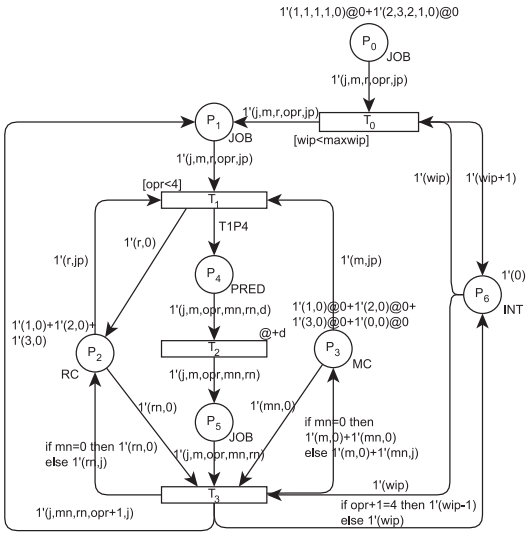

Fig. 3. Modified model of FMS Example 1 using a DL control policy.

by firing the following sequence of transitions from M_0:

$$M_0 \xrightarrow{T_1:j=2,m=3,r=2,\text{opr}=1,jp=0} M_1 \xrightarrow{T_1:j=1,m=1,r=1,\text{opr}=1,jp=0} M_2$$
$$M_2 \xrightarrow{T_2:j=2,m=3,\text{opr}=1,mn=1,rn=3} M_3 \xrightarrow{T_2:j=1,m=1,\text{opr}=1,mn=2,rn=3} M_4$$
$$M_4 \xrightarrow{T_3:j=1,m=1,\text{opr}=1,mn=2,rn=3} M_5 \xrightarrow{T_1:j=1,m=2,r=3,\text{opr}=2,jp=1} M_6$$
$$M_6 \xrightarrow{T_3:j=2,m=3,\text{opr}=1,mn=1,rn=3} M_7 \xrightarrow{T_1:j=2,m=3,r=2,\text{opr}=1,jp=0} M_8$$
$$M_8 \xrightarrow{T_2:j=1,m=2,\text{opr}=2,mn=3,rn=3} M_9 \xrightarrow{T_2:j=2,m=3,\text{opr}=1,mn=1,rn=3} M_{10}$$
$$M_{10} \xrightarrow{T_1:j=2,m=1,r=3,\text{opr}=2,jp=2} M_{11} \xrightarrow{T_2:j=2,m=1,\text{opr}=2,mn=2,rn=3} M_{12}.$$

Marking M_{12} depicts a DL situation in the FMS and the system is said to be in a DL state. The job type J_1 at the load station (token in P_1) cannot be picked up by the free robot because the assigned machine for its first operation is being occupied by another job J_2 in place P_5. Although the robots are available, all the three machines have been blocked by the jobs. The jobs in P_5 remain in a permanent block state since they continually wait to be transferred for their next operations even though the required machines are not processing any job. The system stops evolving since the machines cannot be freed.

To analyze the impact of DL control policies on scheduling performance, we implement a least-cost strategy that controls the flow of jobs [55], [56] in order to guarantee a DL-free design of the system. It restricts the number of active jobs (work-in-process) that flow within the system simultaneously so as to reduce resource competition. It is based on the notion that the number of DLs increases with the number of active jobs, i.e., the higher the number of jobs introduced, the higher the number of DLs that will be generated. Fig. 3 shows the DL prevention model of FMS Example 1. To implement this policy, two new places (P_0 and P_6) and one transition (T_0) are added to the original model in Fig. 2. Transition T_0 describes the event that controls the release of jobs. The variable *maxwip* is a parameter that determines the maximum number of jobs allowed in the system, as a condition to guarantee DL-free operations. It can be used as a constraint to completely avoid DLs or reduce the number of DLs depending on the value introduced. New jobs are not released to the system until the color value *wip* is below the threshold *maxwip*, i.e., the guard

condition $wip < maxwip$ is satisfied. The value of wip is decreased by one each time a job exits the system, as shown in the output arc expression $T3P6$.

III. TCPN Scheduling of FMS Using State Space Exploration

Once the TCPN of the FMS is specified, the different configurations of the FMS can be simulated by generating its reachability graph. As such, an optimal DL-free schedule can be obtained by finding the optimal path from the initial marking M_0 to the goal marking M_f. The goal marking syntax (without the time stamps) for the TCPN in Fig. 2 can be represented as: $M_f = P_1\ 2'(*,*,*,4,*);\ P_2\ 3'(*,0);\ P_3\ 4'(*,0);\ P_4$ empty; P_5 empty; (* means any color value). This means that all the jobs must have completed their operations ($opr = 4$ in place P_1), the robots and machines are free (P_2 and P_3), and there are no jobs being processed or waiting to be unloaded from any machine (P_4 and P_5).

The reachability set of a TCPN $R(\text{TCPN}, M_0)$ comprises the set of all possible markings reachable from the initial marking M_0 by executing any sequence of enabled transitions. If a marking M is in the reachability set $R(\text{TCPN}, M_0)$, it means that there exists a firing sequence transforming M_0 into M. The timed state space of a TCPN [57] can be represented as a directed graph $TSS = (V, E, M_0)$, where V is the set of nodes, E is the set of directed edges or arcs $E = \{(M, t, M')_{\tau_k} \in V \times (T \times \mathbb{R}) \times V | M[t\rangle_{\tau_k} M'\}$ and M_0 is the initial marking. The nodes in the TSS represent markings (the states of the system) and include the global clock and time stamps [53]. A marking $M' \in V$ is a successor of marking $M \in V$ if $(M, t, M')_{\tau_k} \in E$. We say M' is reachable from marking M. Expanding a marking involves the computation of its successors. A visited marking M is a marking that has been expanded or encountered for the first time. A path between two markings M_0 and M_n is a sequence of markings $\sigma = M_0[t_0\rangle, M_1[t_1\rangle, \ldots, M_{n-1}[t_{n-1}\rangle, M_n$ connected by a sequence of edges with enabling times such that $\forall i \in [0, n-1]$, $(M_i, t_i, M_{i+1}) \in E$.

In TSS exploration, timed transitions are fired depending on the earliest possible time of enabling either as an earliest time state space (ESS) class or a reduced earliest time state space (RSS) class [57]. The ESS class offers an event-driven solution by generating new markings in an untimed manner without considering time constraints, whereas RSS enforces time constraints in which the firing of one transition may disable the others. RSS is used to resolve transition conflicts when a marking enables more than one transition at different times.

For the optimization of production scheduling problems, RSS does not always guarantee optimality since it restricts the number of markings to be explored [58]. Also, RSS may fail to return a feasible solution where there are a high number of potential DLs [59]. Owing to the limitations of RSS, this paper considers the ESS. It has been used in TCPN-based scheduling methods proposed in [50], [60], and [61].

While constructing the state space, the search algorithm verifies whether or not a marking has been previously generated. There are situations where the execution of different sequences of transitions from the initial marking leads to the same marking called *duplicate*. For example, consider the two markings $M_1 = P1\ 1'(1,2)@5 + +1'(2,4)@7;\ P2\ 1'(1)@2 + +1'(2)@4 + +1'(3)@8$ and $M_2 = P1\ 1'(1,2)@3 + +1'(2,4)@4;\ P2\ 1'(1)@7 + +1'(2)@2 + +1'(3)@8$. Both are practically the same except that the time stamps of the tokens are different. In the ESS duplicate detection procedure, the time stamps are carried over into the state space. As such, they will be listed as two unique nodes. Exploring the duplicate marking will lead to the same future behavior with differing time evolutions and could even lead to an infinite exploration in some systems [50].

We consider the CSS method [50], [61], [62] that explores the state space in a compact form by separating the evaluation of the untimed marking from the time stamps. The CSS factors out the notion of time for duplicate detection. A unique marking M is differentiated from another marking M' by comparing $M(M_u)$ and $M'(M'_u)$. Using the CSS approach, it is quite attractive to obtain a finite state space but it becomes more complicated when used for optimization purposes due to the varying time stamp of each equivalent token. It faces a difficult task of selecting the marking with the best time stamp set that would optimize the path to the optimal solution. References [60] and [61] select the most promising marking by comparing the time stamps of each token to determine the marking with lower time stamps. This process may become time-consuming if the number of tokens involved is quite large. We use the criterion described in [50] that employs the actual path cost to reach a marking M from the initial marking M_0 denoted as $g(M)$.

For the CSS procedure $CSS(g(M_1), g(M_2))$, there exist three possible scenarios depending on the $g(M)$ values of the stored marking M_1 and the newly generated duplicate marking M_2.

1) $g(M_1) < g(M_2)$*:* Marking M_2 is discarded since its time stamps may not lead to a better goal marking.
2) $g(M_1) > g(M_2)$*:* The time stamp set of M_1 is replaced. If M_1 has been expanded, M is reevaluated and its previous descendants are pruned to maintain time consistency in the state space.
3) $g(M_1) = g(M_2)$*:* First, the time stamps of the two markings are compared for equality. M_2 is discarded if the time stamp set matches. Otherwise, an additional criterion is required to determine the better marking. Tie breaking can be performed using the global clock values.

A CSS can be used to alleviate the state explosion problem but it is still not a guarantee to prevent the untimed state space from exploding. The proposed anytime approach focuses on the condensed ESS as the underlying search graph.

IV. Anytime Heuristic Search Algorithm for DL-Free Scheduling

First, this section introduces the baseline search algorithm (BFIDA*) and its limitations. Then, we present the anytime search algorithm used to generate the condensed ESS of TCPN models to obtain optimal or near-optimal DL-free schedules with respect to makespan as the performance criterion.

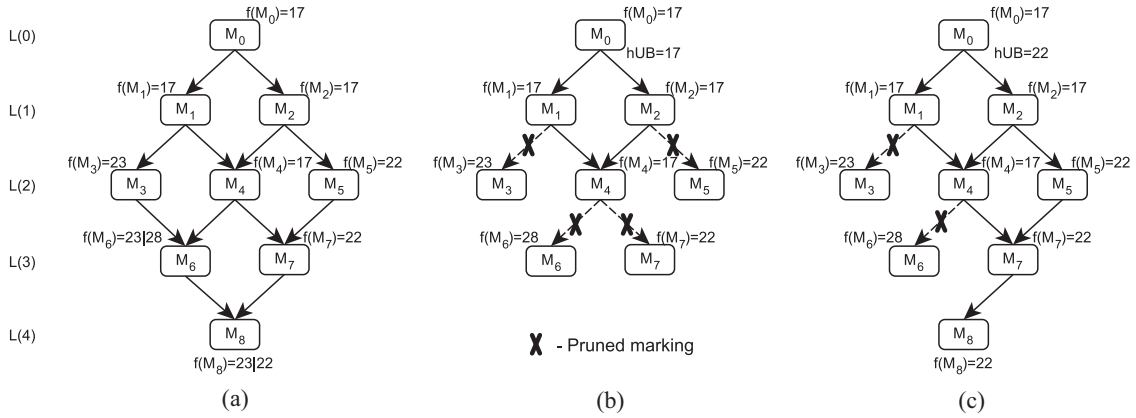

Fig. 4. (a) Condensed ESS of the 2 × 2 job shop. (b) and (c) BFIDA* iterations 1 and 2, respectively.

A. BFIDA*

Typically, A* generates and stores all the successors of an expanded marking, some of which are never expanded since they have a cost value greater than the optimal schedule. These markings fill up the memory and may prevent the search from reaching a goal marking. BFIDA* [52] reduces the memory requirements of A* by performing repeated iterations of breadth-first branch and bound searches (BFBnB) controlled by successively increased cost bound. Each iteration is guided by the same evaluation function (f-cost) as A*; $f(M) = g(M) + h(M)$ where $h(M)$ is a heuristic function that estimates the lower bound on the cost of the best path from M to a goal marking M_f. Unlike A*, the idea is to avoid storing markings that are not likely to lead to an optimal solution at the expense of repeated search effort. At each iteration, it prunes markings with an f-cost that exceeds a hypothetical upper bound (hUB).

BFIDA* starts the iterative search using $f(M_0)$ as the initial hUB. The search terminates successfully if a goal marking is found. Otherwise, BFIDA* restarts the search by increasing hUB by one or to the minimum f-cost of all unexpanded markings from the previous iteration, until a solution is found. The search guarantees that the first solution obtained is optimal provided that the heuristic function $h(M)$ is admissible, i.e., it does not overestimate the cost to the goal marking. Each successive iteration of BFIDA* expands a larger number of markings than the previous iteration, but it never regenerates a marking in the same iteration. In the last iteration, BFIDA* expands the same number of markings as A*. Also, it removes the overhead of maintaining a priority queue.

The admissible heuristic function used in [63] has been adapted: $h(M) = \max_i\{\xi_i(M), i = 1, 2, \ldots, N\}$, 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 when the current system marking is represented by M. N is the total number of resources. For FMS with routing flexibility, this heuristic function is not applicable. It is incomplete and may become inadmissible. Because of the combinatorial effects of alternative routings, we set $h(M) = 0$, i.e., $f(M) = g(M)$, which assumes that no heuristic information is available to maintain admissibility. The same function has been used in [1] and [27].

To illustrate how the BFIDA* search works, Fig. 4 shows its application to the condensed ESS graph of the 2 × 2 job shop TCPN given in [61]. The marking descriptors are omitted for clarity. A marking with more than one f-cost value indicates a marking class with different time stamp sets. The BFIDA* search requires two iterations to obtain the optimal schedule [Fig. 4(b) and (c)]. In the first iteration, the search stops at level $L(2)$ since $f(M_6)$ and $f(M_7)$ is greater than $f(M_0) = 17$. The second iteration set hUB = 22, the $f(M_5)$ from the previous iteration. The search terminates at this iteration since a goal marking of $f(M_8) = 22$ is obtained. The peak memory requirement for BFIDA* is 7, whereas A* stores nine markings. A* has to keep markings M_3 and M_6 in the state space even if they will not be expanded. The reduction becomes significant for a larger instance, especially for problems with large branching factors.

However, BFIDA* has four major weaknesses.

1) It has reexpansion overhead. Each time the search is restarted, the markings in the previous iteration must be expanded to regenerate the pruned markings.
2) It can take a long time to return the optimal schedule if the distance of the initial hUB to the optimal solution is quite big such that hUB increases by a small margin resulting in a large number of iterations.
3) It does not return a solution until it reaches an iteration where hUB = optimal schedule. Also, no solution is returned in a situation where the memory required to reach a goal marking exceeds the available memory.
4) The expansion of markings proceeds in a BFS order, i.e., all the markings in a layer must be fully expanded before moving to the next one. The time to reach a goal marking becomes more expensive if the number of markings in each layer with $f(M) \leq$ hUB is quite large.

The proposed ALS algorithm overcomes these limitations, with a focus on DL-free schedules.

B. ALS DL-Free Algorithm

To avoid restarting the search each time the minimum f-cost (hUB) is increased, the proposed ALS algorithm favors a forward or depth search, thereby reducing the time to return a solution. The ALS combines BFIDA* with suboptimal BFHS (sBFHS) [52] and backtracking. It relies on the

layered structure of the breadth-first search (BFS) in which the state space graph is partitioned into a set of levels or layers, $V = \bigcup_{i=0}^{d_{\max}} L(i)$ where $d_{\max}$ is the maximum depth of the optimal solution path (the number of layers required to reach the optimal solution). A layer $L(i)$ comprises the set of markings with an exact distance of i (the level index) from M_0. Like A*, ALS maintains two lists: 1) open and 2) closed. The open list (OPEN) is implemented as a priority 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 expanded (visited) markings.

The ALS performs a series of BFIDA* and sBFHS iterations with two upper bounds, hUB and suboptimal upper bound sUB. In the first iteration, it starts with BFIDA* using $f(M_0)$ as the initial hUB. Here, hUB is used only for expanding markings with $f(M) \leq$ hUB and it does not prune the markings with $f(M) >$ hUB. This is to avoid the reexpansion overhead of BFIDA*. If the search is unable to find a goal marking with the first hUB, i.e., BFIDA* expansion stops at a depth d_{frontier} called the frontier layer, it continues to expand markings from $d_{\text{frontier}+1}$ with sBFHS using sUB as the upper bound for marking expansion until a goal marking is reached. The initial sUB is set to infinite since sBFHS has no prior information of the search space. Then, sUB is updated with the current best $f(M_f)$. The search terminates if sUB is less than or equal to the hUB of unexpanded markings in OPEN between $d = 0$ and d_{frontier}. At this point, the search is said to have converged and an optimal solution has been reached. Otherwise, it backtracks to the deepest layer having the newer minimum f-cost hUB to start another iteration. The search continues to find improved solutions until an optimal goal marking is reached.

In a nutshell, the ALS divides the state space into two parts: an upper segment from layer $L(0)$ to $L(d_{\text{frontier}})$ in which markings are expanded according to BFIDA* with hUB and a lower segment from $L(d_{\text{frontier}+1})$ to $L(d_{\max})$ where only markings with $f(M) <$ sUB are considered for expansion. A similar approach was adopted by [64] in which depth-first search is used as an extended search with IDA*. The frontier layer d_{frontier} is the last depth of the BFIDA* search area in the current iteration where the markings have at least a successor M with $f(M) >$ hUB. The upper bound hUB is the minimum f-cost of the markings in the OPEN list of the BFIDA* area from layer $L(0)$ to $L(d_{\text{frontier}})$ while sUB is the current best solution obtained from sBFHS and it is updated each time a goal marking with a better $f(M_f)$ is found. Like BFHS [52], sBFHS expands markings in a BFS order and uses the same heuristic function as A*. However, sBFHS only considers the markings at the frontier of the BFIDA*, i.e., it starts constructing its own search space from $d_{\text{frontier}}+1$ and does not remove layers from memory.

To reduce the memory requirements, ALS performs admissible pruning using sUB. It periodically prunes the state space when the incumbent sUB is improved. ALS removes markings with $f(M) \geq$ sUB in the two search areas to avoid keeping a large number of unexpanded markings that would not lead to an optimal solution. Although there is a time overhead associated with this process [46], the algorithm has more control over memory usage. Notwithstanding, the pruning process can be delayed until memory is running low.

Algorithm 1 ALS Algorithm

```
1: procedure ALS(TCPN, M_0, M_g, CPU_limit)
2:     i ← 0, d_frontier ← ∞, d_max ← ∞,
3:     g(M_0) ← 0, f(M_0) ← h(M_0)
4:     hUB ← f(M_0), sUB ← ∞
5:     OPEN[i] ← {M_0}, CLOSED ← {M_0, i}
6:     while hUB < sUB and CPU_limit is not exceeded do
7:         SEARCH(i, d_frontier, d_max, hUB, sUB)
8:         hUB ← minimum f-cost of OPEN[k], ∀k ∈ [0, d_frontier]
9:         i ← backtrack to the deepest layer with hUB and update
           d_frontier
10:        if IsImproved(sUB) then
11:            Prune all M ∈ OPEN[k], ∀k ∈ [0, d] : f(M) ≥ sUB
12:        end if
13:    end while
14:    return M_f, sUB and solution path if sUB ≠ ∞ else return
       no solution
15: end procedure
```

Algorithm 1 gives the pseudocode of the ALS algorithm. The required inputs are the TCPN structure, the initial marking M_0, the defined goal marking M_g, and the CPU time limit $\text{CPU}_{\text{limit}}$. The f-cost of M_0 is computed and the parameter hUB keeps track of the minimum f-cost of unexpanded markings in the OPEN list across the layers in the BFIDA* area (line 8) as one of the conditions for solution convergence (line 6). The OPEN and CLOSED lists are indexed by layer index i (line 5).

Each iteration of the search algorithm runs within the while loop (lines 6–13) and calls the SEARCH procedure in Algorithm 2 to explore all the markings whose f-cost does not exceed UB (line 19) given i, d_{frontier}, $d_{\max}$, hUB, and sUB as inputs. The search starts with BFIDA* without pruning, setting the upper bound UB = hUB (line 17) and ends with sBFHS provided the BFIDA* search does not terminate with a goal marking. It switches search at lines 49–52 and updates d_{frontier} accordingly.

Both search methods have the same marking generation algorithm (line 29) based on ESS and expands only markings with $f(M) <$ UB (line 19). When new markings are generated, the algorithm checks whether or not M' is a goal marking. In case it is a goal marking, it performs a second check to determine if its f-cost is better than sUB. Each time the schedule is improved, sUB is updated and a path from the new solution is constructed back to the initial marking (line 32). For a nongoal marking, its f-cost is compared with sUB. The marking is pruned admissibly if $f(M') \geq$ sUB. Otherwise, duplicate detection is performed against the CLOSED list (lines 38–43). The algorithm adds the new marking to OPEN[i+1] and CLOSED provided the untimed marking $M'(M'_u)$ does not already exist. A duplicate untimed marking goes through the CSS procedure $CSS(g(M_{\text{stored}}), g(M'))$ (line 42) described in Section III to determine whether the time stamp set of the already stored marking must be replaced, given the $g(M)$ of the two markings.

Each time the BFIDA* extends the frontier, d_{frontier} is updated and the initial layer of the sBFHS is also updated.

Algorithm 2 BFIDA* and sBFHS Algorithm

16: **procedure** SEARCH(i, $d_{frontier}$, $d_{\max}$, hUB, sUB)
17: UB $\leftarrow$ hUB
18: **while** $OPEN[i] \neq \emptyset$ **do**
19: **for all** markings $M \in OPEN[i] : f(M) \leq UB$ **do**
20: $OPEN[i] \leftarrow OPEN[i] \setminus \{M\}$
21: **if** *NoEnabledTransitions*(M) and *IsNotGoal*(M) **then**
22: $M_a \leftarrow Parent(M)$
23: **while** $NumSuccessor(M_a) = 1$ **do**
24: $CLOSED \leftarrow CLOSED \setminus \{M\}$
25: $M_a \leftarrow Parent(M_a)$
26: **end while**
27: Mark M_a as potential deadlock path and store transition binding
28: **else**
29: **for all** enabled transitions $t \in T : M[t\rangle_{\tau_k} M'$, $\nexists \tau_{k'} < \tau_k : M[t\rangle_{\tau_{k'}}$ **do**
30: **if** $IsM_g(M')$ **then**
31: **if** sUB $> f(M')$ **then**
32: sUB $\leftarrow f(M')$, $M_f \leftarrow M'$, construct solution path
33: **end if**
34: **if** $d_{\max} = \infty$ or $d_{\max} > i+1$ **then**
35: $d_{\max} = i+1$
36: **end if**
37: **else if** $f(M') <$ sUB **then**
38: **if** $M'(M'_u) \notin CLOSED$ **then**
39: $CLOSED \leftarrow CLOSED \cup \{M', i+1\}$
40: $OPEN[i+1] \leftarrow OPEN[i+1] \cup \{M'\}$
41: **else**
42: $CSS(g(M_{stored}), g(M'))$
43: **end if**
44: **end if**
45: **end for**
46: **end if**
47: **end for**
48: $i \leftarrow i+1$
49: **if** $f_{\min}(OPEN[i]) >$ hUB **then**
50: $UB \leftarrow$ sUB $- 1$
51: $d_{\text{frontier}} \leftarrow i-1$
52: **end if**
53: **end while**
54: **end procedure**

As d_{frontier} is increased in subsequent iterations, the previously generated layers in the sBFHS area merge with the upper area to form part of the BFIDA* region. When a layer from the sBFHS search space is moved into the BFIDA* area, the expansion order changes according to hUB. The previously generated markings eligible under sUB are only selected for expansion if their f-cost values are less than or equal to hUB. Depth $d_{\max}$ is updated each time a goal marking with an improved solution (of lesser number of transition firings) is reached (lines 34–36). This is to ensure that the search is always confined to the shortest path depth. The ALS algorithm terminates when the two upper bounds converge or the search exceeds the given time limit $\text{CPU}_{\text{limit}}$ (line 6).

A DL detection phase is incorporated into the algorithm when markings are selected for expansion from the OPEN list (lines 21–27). It determines whether a marking is deadlocked or not and steps are taken to avoid a repeated occurrence of such a situation, using the information derived from the transition bindings. A marking is detected as a DL if no transition is enabled and the path leading to the marking is no longer explored. Keeping all dead markings in the CLOSED list may lead to a quick state explosion if there are many DLs in the state space [29]. However, they are no longer relevant to obtain a schedule. Since the reachability graph is explored in fragments and guided by a heuristic function, not all DLs are encountered (or selected) by the search algorithm.

In the ALS algorithm, dead markings are either pruned by this procedure (lines 21–27) when selected or removed from the state space when their f-cost values exceed sUB. A selected dead marking is pruned from the CLOSED list as well as its immediate ascendants with one successor on the path to DL. The pruning stops until it gets to an ascendant marking M_a having more than one successor (lines 23–26). Then, M_a is marked as a potential DL marking with forbidden paths, and the transition binding that triggers the generation of the DL path is stored. The stored binding is used as a look-ahead option to avoid the repeated exploration of a previously generated DL path. This is necessary when the same DL path is reachable from different markings in the state space or when a marking with better $g(M)$ is reached in subsequent iterations.

ALS is equivalent to BFIDA* without pruning if a goal marking is reached at the first iteration. The ALS algorithm is complete and optimal provided that:

1) $h(M)$ is a lowerbound on the cost to goal, i.e., $h(M) \leq h^*(M)$, $\forall M$ where $h^*(M)$ is the real optimal remaining cost;
2) $g(M)$ does not discard markings leading to an optimal solution in the time stamp evaluation for CSS;
3) the two upper bounds converge before memory runs out or before the search is terminated.

A good anytime behavior of the algorithm depends on the number of markings at the frontier and the degree of the branching factor. The larger the branching factor and the number of markings at d_{frontier}, the longer the time it takes to reach a solution. To provide timely suboptimal solutions, especially for problems with large branching factors, we propose to limit the number of markings to be expanded at each layer to a given width value ω in each iteration, i.e., the most promising ω markings. ω is called the expansion width and is similar to the beam width [65], [66]. The significant difference between ω and beam width is that ω is not oriented toward satisfying memory constraints but rather to enhance time-efficient solutions. Also, it does not prune the remaining markings not selected for expansion unlike the beam width. Since the search is guided by cost functions, DL markings can be given priority over the nonDL ones. This can lead to a wrong selection of potential DL markings at the early decision stages in the state space, thereby increasing the time overhead. As such, the expansion width must be large enough to avoid frequent backtracking.

Fig. 5 shows the ALS exploration of the condensed ESS in Fig. 4. The scheduling problem is solved in the first iteration. The BFIDA* search stops at $d_{\text{frontier}} = 2$ since the successors of marking M_4 have f-cost values greater than the initial hUB $= 17$. Then, the sBFHS extends the search from layer $L(3)$ and obtains a new solution sUB $= 22$ when the goal marking M_8 is reached. The search terminates without backtracking as sUB is equal to the next hUB value ($f(M_5)$).

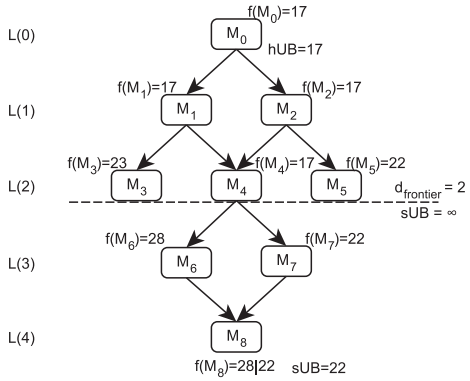

Fig. 5. ALS exploration of the condensed ESS in Fig. 4.

V. Computational Results

The performance of the proposed ALS algorithm is evaluated on three application examples. The first example has been described in Section II, the second is an FMS taken from a real automated manufacturing workstation [24], [25], while the last example is a well studied FMS problem with flexible part routing and multiple resource capacity [7]–[9], [19], [31]. Five different cases of DL-prone situations are considered to investigate the impact of resource competitions and limited buffer capacity on FMS scheduling.

ALS is compared with BFIDA* and other existing methods on DL-free scheduling. The two algorithms described in this paper (ALS and BFIDA*) and the TCPN simulator used for computing transition firings were implemented using C++. The graphical structures of the models are defined in ASCII files whose syntax follows the standard rules of the TCPN formalism. The experiments were conducted on a 2.60 GHz AMD Opteron processor PC with 4 GB RAM. We set $\text{CPU}_{\text{limit}}$ at 3600 s. The performance measures considered are: the last hUB value in the state space at termination, the number of markings expanded *N*mark, the optimal or best makespan returned C_{sol}, the CPU time (in s) and the SQ. SQ is measured between 0 and 1 as C_{best}/C_{solT} where C_{best} is the best-known makespan and C_{solT} is the makespan of the solution returned at time T within the time slot given. For each table, we showed the makespan of the first solution C_{sol-1} and the time it was obtained CPU_{sol-1}. Also, the solutions obtained at different CPU time intervals over the time-limit horizon were shown except in those cases where the best solutions converged at the first iteration.

Example 2: This example considers the scheduling of an automated manufacturing workstation, which forms a subset of an FMS shop floor control system [24], [25]. The system is composed of three machines M_1, M_2, and M_3, a robot R, and a part load/unload station. The robot is responsible for loading/unloading and handling the transportation of parts from one machine to another. There are four job types J_1, J_2, J_3, and J_4 to be processed. It is assumed that jobs come into the workstation on a transport device like a cart or an automated guided vehicle which moves out as soon as the job is picked up by the robot. In a similar way, carts move out from the exit point as soon as they are loaded. The production

TABLE IV
Processing Sequence and Operation Times

Jobs	Machine processing time			Transport operation time			
	1	2	3	1	2	3	4
J_1	M_1:40	M_2:100	M_3:36	5	3	5	4
J_2	M_2:45	M_1:65	M_3:98	5	3	6	4
J_3	M_1:212	M_2:73	M_3:32	6	7	4	5
J_4	M_3:55	M_2:65	M_1:35	4	3	5	5

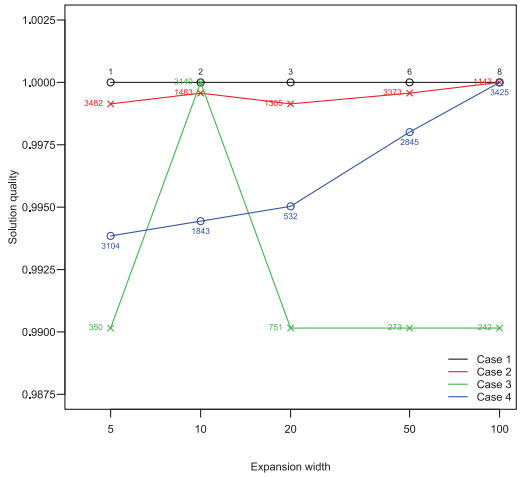

Fig. 6. SQ comparison of expansion width values ($\omega = 5, 10, 20, 50,$ and 100) for all cases of lot size 10 for each job. Inscriptions of CPU time at each SQ point.

sequence and processing times of jobs and transportation times of the robot are given in Table IV. Each job has seven operations in total: three machining and four handling operations performed by the robot. The transfer times of the robot are sequence-independent. Transportation times 1 and 4 represent the loading and unloading time from and to the station, respectively, while transportation times 2 and 3 represent the movement of the jobs for the second and third machining operation, respectively.

The TCPN of the workstation is similar to that of Example 1. The model is not shown here due to space constraints. Place P_2 is converted to a timed place that adds the robot time stamp in the scheduling process. Additional untimed places P_6 and P_7 are attached to T_1 and T_3 respectively. They provide the movement times of the robot for loading/transporting (P_6/T_1) and unloading (P_7/T_3) operations. T_1 and T_3 become timed transitions with assigned durations. We extended the model to two cases of finite buffer capacity and no MHS (assuming MHS tasks are ignored).

In all cases, ALS was executed without ω for lot sizes (of each job) between 1 and 5 classified as low branching factor instances. For large lot sizes ($\geq$ 10), we considered the selection of an appropriate width on a case-by-case basis using the lot size of 10 as a sampling instance. Each instance is tested with varying ω between 5 and 100. It has already been demonstrated in [67] that SQ does not always monotonically improve with an increase in width size. Based on the sampling performance, a width value is then chosen taking into account SQ-time trade-off. Fig. 6 shows the comparison

TABLE V
CASE 1—SCHEDULING RESULTS FOR DIFFERENT LOT SIZES

Lot sizes		BFIDA*				ALS					HHS	AA	
						First solution		Optimal/best solution (CPU=3600)					
J_1	J_2	Depth	Nmark	C_{max}	CPU	C_{sol-1}	CPU_{sol-1}	Nmark	C_{sol}	CPU	C_{sol}	C_{sol}	CPU
1	1	18	58	22	0.02	29	0.00	69	22*	0.02	22	22	1
2	2	36	259	39	0.11	46	0.05	364	39*	0.14	39	39	2.6
5	5	90	1,420	90	0.66	97	1.06	3,680	90*	1.70	90	90	8.1
10	10	180	5,035	175	2.54	182	0.34	1,494	175*	0.72	175	175	19
20	20	360	18,565	345	10.11	352	0.76	3,074	345*	1.62	351	___	___
50	50	900	109,555	855	70.86	862	2.36	7,814	855*	4.96	___	___	___
100	100	1800	429,205	1702	352.51	1712	6.13	15,714	1705*	12.76	___	___	___

of different width performances for the first four cases with CPU inscriptions.

A. Case 1—No Buffer Storage With Multiple Shared MHS

Let us consider Example 1: the three robots as shared resources are used to move parts from one machine to the other without intermediate buffer storage. Note that the transportation time is neglected in this case. The algorithms' performance is examined on the TCPN model with increasing lot sizes. In Fig. 6, the SQ graph is constant over the different widths for case 1 while the CPU time is not improved as the width increases. All the solutions converged before the time limit irrespective of the width. As such, a small width value $\omega = 5$ is sufficient to explore the search graph for large lot sizes.

Table V presents the DL-free scheduling results obtained for lot sizes of $1, 2, 5, 10, 20, 50$, and 100 (for each job type). Besides BFIDA*, ALS performance was benchmarked against two prior works: a PN-based hybrid heuristic search (HHS) that combines best-first search with controlled backtracking [28] and an A* search based on AA formalism that employs supervisory control theory to synthesize DL-free models for scheduling [27]. Note that computation times were not reported for the HHS. The asterisk (*) in the result table indicates an optimal schedule in which the makespan converged before the specified time limit. BFIDA* returned optimal DL-free schedules for all the instances under the available memory but the CPU time increases with the problem size. For the ALS, the optimal schedules are obtained in relatively short computation times even for the lot size of 100. In comparison with BFIDA*, ALS expanded lesser number of markings for larger lot sizes (≥ 10) and needed only three iterations to obtain the optimal schedules for each instance. HHS fails to obtain the optimal schedule for the lot size of 20. Solutions were not reported for lot sizes > 20 in previous works.

B. Case 2—No Buffer Storage Without MHS

Using Example 2, this case assumes that parts can move automatically from one machine to the next one without MHS and no buffer storage for work-in-process [24]. The width $\omega = 100$ is selected given the SQ comparison profile in Fig. 6 for case 2. It returns the best solution within the first 20 min. The anytime performance profiles in Fig. 7 showed that smaller width values quickly return a stream of improving schedules for the first 2 min whereas larger width values (≥ 50) start off with higher quality solutions requiring more computational effort. Although the smaller widths needed more time to reach the same SQ obtained by the larger ones.

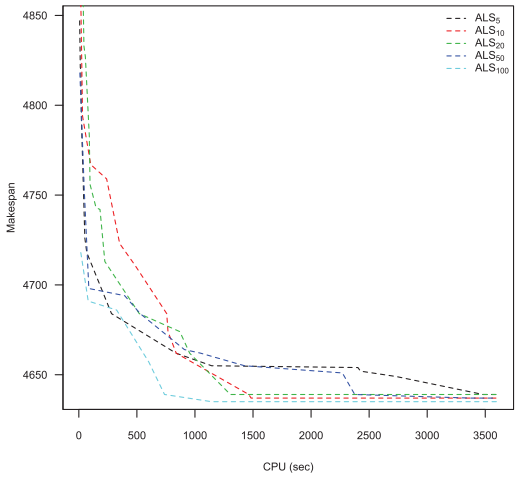

Fig. 7. Case 2—Anytime performance comparison for different expansion width values of lot size (10, 10, 10, 10).

Table VI reports the DL-free scheduling results obtained for several different lot sizes. As expected, ALS expanded lesser number of markings and needed short computation time when compared with BFIDA*. BFIDA* took over 18 h to obtain the optimal schedule for lot size (5, 5, 5, 5) which is unacceptable for a moderate instance size. On the other hand, ALS only needed 530 s to find the optimal schedule. The best makespan obtained for lot size instances ($\geq$ (10, 10, 10, 10)) did not converge within the first 3600 s. Though the gap between the last hUB values and the best solutions returned is quite large, it is expected to improve if given more time.

C. Case 3—Finite Buffer Capacity Without MHS

Case 3 is an extended version of Example 2 considering finite buffer size (bf) without using the robot to move parts from one machine to the other. An intermediate buffer is placed between two machines with a given capacity. As shown in Fig. 6 for case 3, width value $\omega = 10$ is sufficient

TABLE VI
CASE 2—SCHEDULING RESULTS FOR DIFFERENT LOT SIZES

Lot sizes				BFIDA*				ALS					
									First solution		Optimal/best solution (CPU=3600)		
J_1	J_2	J_3	J_4	Depth	Nmark	C_{max}	CPU	hUB	C_{sol-1}	CPU_{sol-1}	Nmark	C_{sol}	CPU
1	1	1	1	36	9,749	512	2.79	512	533	0.22	1,181	512*	0.30
2	2	0	2	54	59,567	603	19.46	603	639	0.13	2,936	603*	0.45
3	4	1	2	90	2,860,993	1061	1129.46	1061	1061	10.05	36,075	1061*	10.05
5	5	5	5	180	136,301,623	2311	67270.70	2311	2356	200.24	2,212,418	2311*	529.82
10	10	10	10	360	—	—	—	3837	4718	17.02	6,258,173	4635	1143.26
20	20	20	20	720	—	—	—	7256	9438	42.54	5,473,263	9265	3455.39
50	50	50	50	1800	—	—	—	17731	23431	152.80	4,084,656	23364	2946.55

TABLE VII
CASE 3—SCHEDULING RESULTS FOR DIFFERENT LOT SIZES WHERE BUFFER CAPACITY IS 2

Lot sizes				BFIDA*				ALS						HHS
									First solution		Optimal/best solution (CPU=3600)			
J_1	J_2	J_3	J_4	Depth	Nmark	C_{max}	CPU	hUB	C_{sol-1}	CPU_{sol-1}	Nmark	C_{sol}	CPU	C_{sol}
2	2	0	2	54	164,366	455	86.06	455	455	4.06	15,993	455*	4.06	455
5	4	6	3	162	o.o.m	o.o.m	o.o.m	1837	1942	2.98	1,430,778	1837*	1051.21	1942
10	10	10	10	360	—	—	—	3520	3811	20.44	3,749,469	3520*	3139.54	3638
20	20	20	20	720	—	—	—	7040	7331	96.05	3,942,860	7110	457.67	7171
50	50	50	50	1800	—	—	—	17600	19118	651.63	3,395,074	17670	2325.96	—

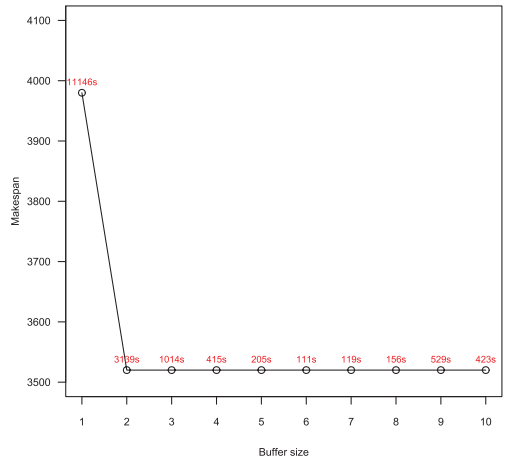

Fig. 8. ALS scheduling results of lot size (10, 10, 10, 10) for different buffer sizes ranging from 1 to 10.

for a good anytime performance profile to explore large lot sizes. Fig. 8 shows the DL-free scheduling results for lot size (10, 10, 10, 10) with buffer sizes ranging from 1 to 10. As seen in the figure, the $bf = 1$ places more restrictions on the system which led to a higher total completion time. Since there is insufficient space for work-in-process parts, the number of DL states increases requiring more computation time to find the best solution. Note that the solution did not converge within the time limit for this experiment. However, an optimal and constant makespan is obtained for bf $\geq$ 2. The computational time to obtain the converged schedule decreases considerably as capacity increases. Although there is a CPU rise from bf $\geq$ 7, the computation time is relatively low when compared with those of buffer sizes between 1 and 3. From the depicted performance, we can conclude that systems that assume infinite buffer capacity are easier to solve.

Table VII presents the scheduling results of different lot sizes (up to 50 for each job type) where $bf = 2$. BFIDA* could not solve lot sizes from (5, 4, 6, 3) since it runs out of memory and fails to return any solution. ALS obtained the optimal schedules for the first three problems while the SQ returned for the other problems is high and the best schedules obtained are quite close to the last hUB. Also, ALS outperformed HHS as HHS could only solve the smallest instance to optimality.

D. Case 4—No Buffer Storage With Single Shared MHS

Case 4 considers Example 2 without extension. This case is more complex than the previous cases as it integrates the movement time of the robot into production scheduling. Based on the anytime performance shown in Fig. 9 and the makespan performance profile in Fig. 6 for different widths, $\omega = 100$ is considered appropriate to achieve a good SQ-time trade-off. In this case, increasing the width actually improves the SQ. Fig. 9 shows that smaller values of ω can considerably reduce solution times while still obtaining high-quality solutions.

Table VIII gives the scheduling results of different lot sizes (up to 50 for each job). ALS obtained the best solutions with much lesser computation times than the other methods. Similar to case 2, it took BFIDA* more than 22 h to reach a goal marking for lot size (5, 5, 5, 5), whereas ALS solved the same problem in 516 s. The AA method in [27] concluded that optimal makespan values have been reached for all the instances solved in their work. However, the optimal schedules obtained by ALS and BFIDA* have demonstrated otherwise. This implies that the integration of DL controllers

TABLE VIII
CASE 4—SCHEDULING RESULTS FOR DIFFERENT LOT SIZES

Lot sizes				BFIDA*				ALS						AA	
									First solution		Optimal/best solution (CPU=3600)				
J_1	J_2	J_3	J_4	Depth	Nmark	C_{max}	CPU	hUB	C_{sol-1}	CPU_{sol-1}	Nmark	C_{sol}	CPU	C_{sol}	CPU
1	1	1	1	36	30,814	560	9.70	560	643	0.09	1,021	560*	0.25	560	2.3
2	2	0	2	54	150,836	660	53.62	660	660	0.75	3,065	660*	0.75	665	3.2
3	4	1	2	90	3,314,356	1157	1391.72	1157	1157	8.53	33,601	1157*	8.53	1244	12.1
5	5	5	5	180	127,064,461	2503	81522.80	2503	2535	133.01	1,051,727	2503*	516.21	2513	175
10	10	10	10	360	—	—	—	3985	5119	17.71	6,201,302	5008	3425.08	—	—
20	20	20	20	720	—	—	—	7402	10185	42.04	5,558,489	10006	1152.11	—	—
50	50	50	50	1800	—	—	—	17708	25299	145.06	4,187,435	25014	3451.30	—	—

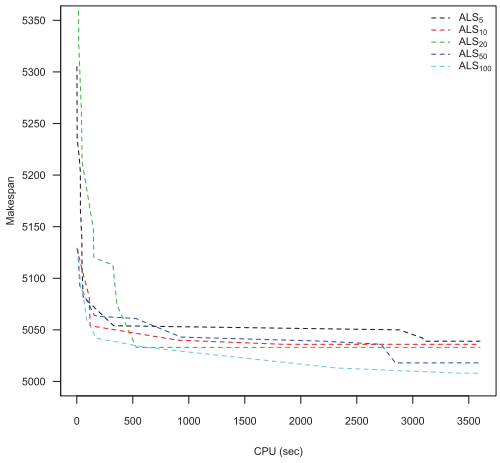

Fig. 9. Case 4—Anytime performance comparison for different expansion width values of lot size (10, 10, 10, 10).

does not always guarantee optimality even when the control policies are considered to be optimal as in [27]. The actual quality of the best makespan returned for lot sizes ($\geq$ 10) cannot be reported. The hUB may not be a good parameter to measure how close the solutions are to optimality since it is a hypothetical bound that depends on the evolution of the cost values in the state space.

E. Case 5—Routing Flexibility With Multiple Resource Capacity

Case 5 is the third FMS example [7]–[9], [19], [31] that consists of four machines M_1, M_2, M_3, and M_4 and three robots R_1, R_2, and R_3. Each machine can hold two parts at the same time while the robots can move only one part at a time. Parts enter the system through three loading buffers $I1$, $I2$, and $I3$ and leave the system through three unloading buffers $O1$, $O2$, and $O3$. No intermediate buffers exist in the system. Three types of jobs J_1, J_2, and J_3 are produced. The operation sequence (and processing times) for job J_1 is $R_2(8), M_2(34)$, and $R_2(5)$ with three operations while job J_3 has five operations, $R_3(5), M_4(22), R_2(4), M_3(17)$, and $R_1(6)$. Job J_2 has two different processing routes; $R_1(4), M_1(32), R_2(8), M_2(38), R_2(5)$ or $R_1(4), M_3(23), R_2(6), M_4(20), R_3(5)$. The TCPN model is not described in this paper due to space considerations.

The ALS algorithm is tested on 16 instances of the system proposed by [31]. It contains four instance sets of different resource capacities (numbers of machines $C(M_i) : i \in \{1, 2, 3, 4\}$ and robots $C(R_i) : i \in \{1, 2, 3\}$ in each type), where each set has four instances of different multiple lot sizes in the order (8, 12, 8), (10, 20, 10), (15, 20, 15), and (20, 20, 20) for each job type J_1, J_2, and J_3 respectively.

1) *In*01–*In*04*:* The original system configuration with $C(M_i) = 2$ and $C(R_i) = 1$.
2) *In*05–*In*08*:* $C(M_i) = 2$ and $C(R_i) = 2$.
3) *In*09–*In*12*:* $C(M_i) = 3$ and $C(R_i) = 2$.
4) *In*13–*In*16*:* $C(M_i) = 3$ and $C(R_i) = 3$.

ALS performance is compared with the genetic algorithm (GA2) proposed by [31] which embeds five different DL controllers, namely E + GA2, L + GA2, H + GA2, X + GA2, and P + GA2. The BFIDA* algorithm was not considered for this experiment as it could not handle the state space size of the systems due to the large number of parts and resources involved. A width of 10 is used to explore the state space. Table IX shows the performance comparison of the scheduling results obtained using the proposed ALS algorithm and the GA2 method for the 16 instances. For most of the instances, a very good solution was found quickly in less than 5 s and this solution is in each case better than the best solution reported for the GA2 algorithm, especially E + GA2 and L + GA2. ALS provided improved solutions (in bold) for 11 instances out of 16 whereas the X + GA2 performed better in the rest, specifically three of the four instances in the last set *In*13–*In*16. X + GA2 proved to be more efficient in the last instance set. The ALS performance in this set can be attributed to the quality of the heuristic function for a larger capacity of resource set. The results show that solutions were greatly improved in the first 5 min while for few instances like *In*04, *In*11, *In*12, and *In*15, better schedules were periodically found. As expected, increasing the number of resources results in reduced makespans.

F. Comparison Between DL Prevention and DL-Prone Models

The performance of the ALS algorithm for the DL control policy described in Section II is compared with the DL-prone versions of the system. Table X shows the computational results for increasing values of *wipmax* for lot sizes $J_1 = 5, J_2 = 4, J_3 = 6, J_4 = 3$ and $J_1 = 3, J_2 = 4, J_3 = 1$, $J_4 = 2$ in cases 3 and 4, respectively. The DL column in

TABLE IX
COMPARISON OF SCHEDULING RESULTS OBTAINED USING THE PROPOSED ALS ALGORITHM AND GA2 METHOD [31]

Instance	ALS								GA2				
	First solution		Solutions at CPU intervals (minutes)				Best solution (CPU=3600)		E+GA2	L+GA2	H+GA2	X+GA2	P+GA2
	C_{sol-1}	CPU	5	10	20	40	C_{sol}	CPU	C_{sol}	C_{sol}	C_{sol}	C_{sol}	C_{sol}
In01	417	2.12	301	301	298	298	**293**	2436.22	420	416	338	352	331
In02	469	3.57	403	402	402	397	**397**	1637.47	645	644	525	531	475
In03	822	3.96	497	490	___	___	**490**	586.83	760	784	648	717	612
In04	1037	4.31	610	595	595	589	**587**	3057.53	907	895	837	890	803
In05	271	2.57	271	266	___	___	266	403.17	294	319	___	**254**	___
In06	389	3.88	374	368	___	___	**368**	555.78	438	510	___	370	___
In07	493	5.07	453	452	450	___	**450**	853.01	545	601	___	485	___
In08	894	8.69	569	___	___	___	**569**	291.50	629	691	___	602	___
In09	275	2.90	198	196	___	___	196	575.91	219	239	___	**184**	___
In10	288	4.87	277	277	273	269	**269**	1930.60	327	356	___	278	___
In11	509	5.49	347	335	332	___	**332**	1024.22	404	437	___	356	___
In12	663	6.13	439	416	416	409	**409**	1888.93	490	523	___	426	___
In13	228	3.07	193	189	189	187	186	3235.68	191	209	___	**170**	___
In14	288	4.70	282	276	275	___	275	1741.54	285	318	___	**247**	___
In15	493	5.43	367	349	331	327	**327**	1399.34	347	406	___	332	___
In16	413	6.72	407	407	401	396	396	1353.38	418	475	___	**389**	___

TABLE X
PERFORMANCE COMPARISON OF THE PROPOSED ALGORITHM WITH DL PREVENTION AND DL-PRONE MODELS

Instance	Deadlock prevention TCPN				Deadlock-prone TCPN		
	$wipmax$	DL	C_{sol}	CPU	DL	C_{sol}	CPU
C4 3,4,1,2	1	0	2166	1.59	719	1157	8.53
	2	583	1234	12.78			
	3	1352	1157	45.38			
	4	2186	1157	95.69			
	5	2678	1157	145.99			
	6	2117	1157	182.85			
	7	1590	1157	184.72			
	8	1276	1157	159.03			
	9	1142	1157	111.17			
	10	1101	1157	56.52			
C3 5,4,6,3	1	0	2132	14.71	12560	1837	1051.21
	2	0	1907	22.20			
	3	**13565**	1872	98.41			
	4	3715	1837	1379.04			
	5	4743	1837	2207.75			
	6	5923	1837	3009.40			
	7	**6599**	1872	169.37			
	8	5601	1837	3395.02			

the table refers to the number of DLs encountered during the scheduling process and not the total number of DLs that exist in the system. Note that the algorithm did not converge for *wipmax* = 3 and *wipmax* = 7 for case 3. The results show that the SQ produced is quite low when DLs are completely avoided in the model. Thus, DL control policies can be used to reduce the search space, but they do not necessarily enhance the scheduling performance. However, the algorithm performance is improved with respect to the computation time as observed in the table. Han *et al.* [31] already pointed out that the performance of a system can vary depending on the policy adopted since different policies impose different restrictions. This clearly demonstrates the restriction of control policies on the system performance and shows that better scheduling performance can be obtained with DL-prone models using the proposed approach.

VI. CONCLUSION

An anytime heuristic search method has been proposed for the DL-free scheduling problem of FMS with shared resources, modeled as a discrete-event system. It finds optimal or near-optimal DL-free schedules for a given initial marking of the system based on the reachability graph of TCPN modeling. DL are avoided in schedules during the construction of the graph by checking whether or not the sequence of transition firings lead to a marking from which no other transition can be fired. The proposed algorithm overcomes the limitations of conventional heuristic search algorithms. It returns quick solutions and gradually improves the incumbent feasible solution obtained over time until the solution converges or the run-time limit is exceeded. The approach is applicable to time-constrained environments and real-time situations where decisions must be taken at the slightest possible time.

The algorithm has been tested extensively on five different cases of DL-prone situations that take into account limitations that arise in realistic manufacturing systems. Case 5 is the most difficult while cases 2 and 4 proved to be more challenging than cases 1 and 3. Lesser number of DL are encountered with multiple shared resources or when there is sufficient buffer capacity. Also, the inclusion of the robot transportation time as an operation increases the problem complexity. For instances that did not converge within the specified time limit, the time can be extended to allow the algorithm obtain the optimal schedule.

Computational results demonstrate that solutions of high quality can be found quickly (less than 100 s in most cases) with the proposed anytime search algorithm. The best solutions provided in the result tables do not indicate the algorithm's limit but rather its effectiveness in producing solutions of high quality in a limited time frame. None of the experiments ran

out of memory within the time limit. However, it is expected that the search space will exceed the available memory over time. This can be circumvented by parallelizing the anytime search on a number of processors with distributed memory in order to increase the amount of memory available as well as to reduce the time to return improved solutions and solution convergence.

In anytime approaches, it is expected that the search keeps improving the solution at regular time intervals, but this is not definitely true as shown in Figs. 7 and 9 and Table IX. ALS can go on for a long time without finding an improved schedule. One of the possible reasons can be due to the fact that ALS might have reached the optimal schedule unknowingly with the current sUB. Since hUB is increased according to the evolution of minimum $f(M)$ such that the number of iterations can be quite large if there is a big distance between the two. To speed up convergence, it is proposed to increase hUB to the incumbent sUB after an x time period without solution improvement to validate the optimal DL-free schedule. Also, an incremental cost strategy can be implemented to compute a new hUB based on the distance of the hUB from sUB with respect to the depth of the graph. The application of the proposed approach to simultaneous production scheduling and conflict-free routing problems for automated guided vehicles [4] will be investigated in a future work.

REFERENCES

[1] D. Y. Lee and F. DiCesare, "Scheduling flexible manufacturing systems using Petri nets and heuristic search," *IEEE Trans. Robot. Autom.*, vol. 10, no. 2, pp. 123–132, Apr. 1994.

[2] K. Baker, *Introduction to Sequencing and Scheduling*. New York, NY, USA: Wiley, 1974.

[3] G. Mejia and C. Montoya, "Scheduling manufacturing systems with blocking: A Petri net approach," *Int. J. Prod. Res.*, vol. 47, no. 22, pp. 6261–6277, 2009.

[4] T. Nishi, Y. Hiranaka, and I. E. Grossmann, "A bilevel decomposition algorithm for simultaneous production scheduling and conflict-free routing for automated guided vehicles," *Comput. Oper. Res.*, vol. 38, no. 5, pp. 876–888, 2011.

[5] I. Abdallah and H. ElMaraghy, "Deadlock prevention and avoidance in FMS: A Petri net based approach," *Int. J. Adv. Manuf. Technol.*, vol. 14, no. 10, pp. 704–715, 1998.

[6] I. B. Abdallah, H. A. Elmaraghy, and T. Elmekkawy, "Deadlock-free scheduling in flexible manufacturing systems using Petri nets," *Int. J. Prod. Res.*, vol. 40, no. 12, pp. 2733–2756, 2002.

[7] K. Xing, L. Han, M. Zhou, and F. Wang, "Deadlock-free genetic scheduling algorithm for automated manufacturing systems based on deadlock control policy," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 3, pp. 603–615, Jun. 2012.

[8] J. Ezpeleta, J. Colom, and J. Martinez, "A Petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Trans. Robot. Autom.*, vol. 11, no. 2, pp. 173–184, Apr. 1995.

[9] Z. Li and M. Zhou, "Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 1, pp. 38–51, Jan. 2004.

[10] L. Piroddi, R. Cordone, and I. Fumagalli, "Selective siphon control for deadlock prevention in Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 6, pp. 1337–1348, Nov. 2008.

[11] Y. Huang, M. Jeng, X. Xie, and S. Chung, "Deadlock prevention policy based on Petri nets and siphons," *Int. J. Prod. Res.*, vol. 39, no. 2, pp. 283–305, 2001.

[12] H. Hu, M. Zhou, and Z. Li, "Liveness and ratio-enforcing supervision of automated manufacturing systems using Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 2, pp. 392–403, Mar. 2012.

[13] H. Hu, M. Zhou, Z. Li, and Y. Tang, "Deadlock-free control of automated manufacturing systems with flexible routes and assembly operations using Petri nets," *IEEE Trans. Ind. Inf.*, vol. 9, no. 1, pp. 109–121, Feb. 2013.

[14] H. Hu and Y. Liu, "Supervisor simplification for AMS based on Petri nets and inequality analysis," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 1, pp. 66–77, Jan. 2014.

[15] Y. Chen, Z. Li, K. Barkaoui, and M. Uzam, "New Petri net structure and its application to optimal supervisory control: Interval inhibitor arcs," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 10, pp. 1384–1400, Oct. 2014.

[16] H. Liu, K. Xing, M. Zhou, L. Han, and F. Wang, "Transition cover-based design of Petri net controllers for automated manufacturing systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 44, no. 2, pp. 196–208, Feb. 2014.

[17] J. Ezpeleta and L. Recalde, "A deadlock avoidance approach for nonsequential resource allocation systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 1, pp. 93–101, Jan. 2004.

[18] N. Wu and M. Zhou, "Avoiding deadlock and reducing starvation and blocking in automated manufacturing systems," *IEEE Trans. Robot. Autom.*, vol. 17, no. 5, pp. 658–669, Oct. 2001.

[19] K. Xing, M. Zhou, H. Liu, and F. Tian, "Optimal Petri-net-based polynomial-complexity deadlock-avoidance policies for automated manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 1, pp. 188–199, Jan. 2009.

[20] R. Wysk, N. S. Yang, and S. Joshi, "Detection of deadlocks in flexible manufacturing cells," *IEEE Trans. Robot. Autom.*, vol. 7, no. 6, pp. 853–859, Dec. 1991.

[21] T. K. Kumaran, W. Chang, H. Cho, and R. A. Wysk, "A structured approach to deadlock detection, avoidance and resolution in flexible manufacturing systems," *Int. J. Prod. Res.*, vol. 32, no. 10, pp. 2361–2379, 1994.

[22] Z. Li, M. Zhou, and N. Wu, "A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 2, pp. 173–188, Mar. 2008.

[23] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on Petri nets—A literature review," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 437–462, Jul. 2012.

[24] H. H. Xiong and M. Zhou, "Deadlock-free scheduling of an automated manufacturing system based on Petri nets," in *Proc. 1997 IEEE Int. Conf. Robot. Autom.*, vol. 2. Albuquerque, NM, USA, pp. 945–950.

[25] S. Ramaswamy and S. Joshi, "Deadlock-free schedules for automated manufacturing workstations," *IEEE Trans. Robot. Autom.*, vol. 12, no. 3, pp. 391–400, Jun. 1996.

[26] Y. Mati, N. Rezg, and X. Xie, "A taboo search approach for deadlock-free scheduling of automated manufacturing systems," *J. Intell. Manuf.*, vol. 12, nos. 5–6, pp. 535–552, 2001.

[27] H. Golmakani, J. Mills, and B. Benhabib, "Deadlock-free scheduling and control of flexible manufacturing cells using automata theory," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 36, no. 2, pp. 327–337, Mar. 2006.

[28] H. H. Xiong, M. Zhou, and R. Caudill, "A hybrid heuristic search algorithm for scheduling flexible manufacturing systems," in *Proc. 1996 IEEE Int. Conf. Robot. Autom.*, vol. 3. Minneapolis, MN, USA, pp. 2793–2797.

[29] M. D. Jeng, W. D. Chiou, and Y. L. Wen, "Deadlock-free scheduling of flexible manufacturing systems based on heuristic search and Petri net structures," in *Proc. 1998 Int. Conf. Syst. Man Cybern.*, vol. 1. San Diego, CA, USA, pp. 26–31.

[30] H. J. Yoon and D. Y. Lee, "Deadlock-free scheduling of photolithography equipment in semiconductor fabrication," *IEEE Trans. Semicond. Manuf.*, vol. 17, no. 1, pp. 42–54, Feb. 2004.

[31] L. Han, K. Xing, X. Chen, H. Lei, and F. Wang, "Deadlock-free genetic scheduling for flexible manufacturing systems using Petri nets and deadlock controllers," *Int. J. Prod. Res.*, vol. 52, no. 5, pp. 1557–1572, 2014.

[32] X. Gang and Z. Wu, "Deadlock-free scheduling strategy for automated production cell," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 1, pp. 113–122, Jan. 2004.

[33] G. Tuncel and G. Bayhan, "Applications of Petri nets in production scheduling: A review," *Int. J. Adv. Manuf. Technol.*, vol. 34, nos. 7–8, pp. 762–773, 2007.

[34] T. ElMekkawy and H. ElMaraghy, "Real-time scheduling with deadlock avoidance in flexible manufacturing systems," *Int. J. Adv. Manuf. Technol.*, vol. 22, nos. 3–4, pp. 259–270, 2003.

[35] H. Hu and Z. Li, "Modeling and scheduling for manufacturing grid workflows using timed Petri nets," *Int. J. Adv. Manuf. Technol.*, vol. 42, nos. 5–6, pp. 553–568, 2009.

[36] S. Lauzon, J. Mills, and B. Benhabib, "An implementation methodology for the supervisory control of flexible manufacturing workcells," *J. Manuf. Syst.*, vol. 16, no. 2, pp. 91–101, 1997.

[37] A. Ramirez-Serrano, C. Sriskandarajah, and B. Benhabib, "Automata-based modeling and control synthesis for manufacturing workcells with part-routing flexibility," *IEEE Trans. Robot. Autom.*, vol. 16, no. 6, pp. 807–823, Dec. 2000.

[38] T. Y. Elmekkawy and H. A. Elmaraghy, "Efficient search of Petri nets for deadlock-free scheduling in FMSs using heuristic functions," *Int. J. Comput. Integr. Manuf.*, vol. 16, no. 1, pp. 14–24, 2003.

[39] H. Yu, A. Reyes, S. Cang, and S. Lloyd, "Combined Petri net modelling and AI-based heuristic hybrid search for flexible manufacturing systems—Part II. Heuristic hybrid search," *Comput. Ind. Eng.*, vol. 44, no. 4, pp. 545–566, 2003.

[40] T. Nishi and Y. Tanaka, "Petri net decomposition approach for dispatching and conflict-free routing of bidirectional automated guided vehicle systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 5, pp. 1230–1243, Sep. 2012.

[41] T. Nishi and I. Matsumoto, "Petri net decomposition approach to deadlock-free and non-cyclic scheduling of dual-armed cluster tools," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 1, pp. 281–294, Jan. 2015.

[42] K. Xing, M. Zhou, F. Wang, H. Liu, and F. Tian, "Resource-transition circuits and siphons for deadlock control of automated manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 1, pp. 74–84, Jan. 2011.

[43] H. Liu, K. Xing, W. Wu, M. Zhou, and H. Zou, "Deadlock prevention for flexible manufacturing systems via controllable siphon basis of Petri nets," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 3, pp. 519–529, Mar. 2015.

[44] J. Luo, K. Xing, M. Zhou, X. Li, and X. Wang, "Deadlock-free scheduling of automated manufacturing systems using Petri nets and hybrid heuristic search," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 3, pp. 530–541, Mar. 2015.

[45] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[46] E. A. Hansen and R. Zhou, "Anytime heuristic search," *J. Artif. Intell. Res.*, vol. 28, no. 1, pp. 267–297, 2007.

[47] S. Aine, P. P. Chakrabarti, and R. Kumar, "AWA*—A window constrained anytime heuristic search algorithm," in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*, San Francisco, CA, USA: Morgan Kaufmann, 2007, pp. 2250–2255.

[48] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, vol. 16, S. Thrun, L. Saul, and B. Scholkopf, Eds. Cambridge, MA, USA: MIT Press, 2003.

[49] S. Vadlamudi, S. Aine, and P. Chakrabarti, "MAWA*—A memory-bounded anytime heuristic-search algorithm," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 3, pp. 725–735, Jun. 2011.

[50] O. T. Baruwa and M. A. Piera, "Anytime heuristic search for scheduling flexible manufacturing systems: A timed colored Petri net approach," *Int. J. Adv. Manuf. Technol.*, vol. 75, nos. 1–4, pp. 123–137, 2014.

[51] S. Vadlamudi, P. Gaurav, S. Aine, and P. Chakrabarti, "Anytime column search," in *AI 2012: Advances in Artificial Intelligence* (Lecture Notes in Computer Science 7691), M. Thielscher and D. Zhang, Eds. Berlin, Germany: Springer, pp. 254–265.

[52] R. Zhou and E. A. Hansen, "Breadth-first heuristic search," *Artif. Intell.*, vol. 170, nos. 4–5, pp. 385–408, 2006.

[53] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. New York, NY, USA: Springer, 2009.

[54] W. van der Aalst, "Interval timed coloured Petri nets and their analysis," in *Application and Theory of Petri Nets 1993*. Lecture Notes in Computer Science, vol. 691, M. A. Marsan, Ed. Berlin, Germany: Springer, pp. 453–472.

[55] P. Lacomme, A. Moukrim, and N. Tchernev, "Simultaneous job input sequencing and vehicle dispatching in a single-vehicle automated guided vehicle system: A heuristic branch-and-bound approach coupled with a discrete events simulation model," *Int. J. Prod. Res.*, vol. 43, no. 9, pp. 1911–1942, 2005.

[56] A. Caumond, P. Lacomme, A. Moukrim, and N. Tchernev, "An MILP for scheduling problems in an FMS with one vehicle," *Eur. J. Oper. Res.*, vol. 199, no. 3, pp. 706–722, 2009.

[57] C. Lakos and L. Petrucci, "Modular state space exploration for timed Petri nets," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, nos. 3–4, pp. 393–411, 2007.

[58] M. Piera and G. Music, "Coloured Petri net scheduling models: Timed state space exploration shortages," *Math. Comput. Simulat.*, vol. 82, no. 3, pp. 428–441, 2011.

[59] G. Mejia and N. G. Odrey, "An approach using Petri nets and improved heuristic search for manufacturing system scheduling," *J. Manuf. Syst.*, vol. 24, no. 2, pp. 79–92, 2005.

[60] M. Mujica, M. A. Piera, and M. Narciso, "Revisiting state space exploration of timed coloured Petri net models to optimize manufacturing system's performance," *Simulat. Model. Pract. Theory*, vol. 18, no. 9, pp. 1225–1241, 2010.

[61] M. E. Narciso, M. A. Piera, and A. Guasch, "A time stamp reduction method for state space exploration using colored Petri nets," *Simulation*, vol. 88, no. 5, pp. 592–616, May 2012.

[62] S. Christensen, L. M. Kristensen, and T. Mailund, "Condensed state spaces for timed Petri nets," in *Applications and Theory of Petri Nets 2001* (Lecture Notes in Computer Science 2075), J.-M. Colom and M. Koutny, Eds. Berlin, Germany: Springer, pp. 101–120.

[63] H. H. Xiong and M. Zhou, "Scheduling of semiconductor test facility via Petri nets and hybrid heuristic search," *IEEE Trans. Semicond. Manuf.*, vol. 11, no. 3, pp. 384–393, Aug. 1998.

[64] C. Mencia, M. R. Sierra, and R. Varela, "Intensified iterative deepening A* with application to job shop scheduling," *J. Intell. Manuf.*, vol. 25, no. 6, pp. 1245–1255, 2013.

[65] W. Zhang, "Complete anytime beam search," in *Proc. 15th Nat. Conf. Artif. Intell. (AAAI)*, Menlo Park, CA, USA, 1998, pp. 425–430.

[66] R. Zhou and E. A. Hansen, "Beam-stack search: Integrating backtracking with beam search," in *Proc. Int. Conf. Autom. Plan. Sched.*, Monterey, CA, USA, 2005, pp. 90–98.

[67] S. Vadlamudi, S. Aine, and P. Chakrabarti, "Incremental beam search," *Inf. Process. Lett.*, vol. 113, nos. 22–24, pp. 888–893, 2013.

Olatunde T. Baruwa received the B.Sc. degree in electronic and computer engineering from Lagos State University, Lagos, Nigeria, in 2004, the M.Sc. degree in management from Pompeu Fabra University, Barcelona, Spain, in 2007, and the M.Res. degree in industrial informatics from the Autonomous University of Barcelona, Barcelona, Spain, in 2012, where he is currently pursuing the Ph.D. degree from the Logistics and Aeronautics Unit.

His current research interests include discrete-event systems modeled by colored Petri nets, scheduling and optimization of flexible manufacturing systems, AI heuristic search methods, simulation-optimization, and airport logistics.

Miquel Angel Piera received the B.Sc. degree in computer science and the Ph.D. degree from the Universitat Autònoma de Barcelona, Barcelona, Spain, in 1988 and 1993, respectively.

He is currently a Professor of Logistics and Aeronautics and the Director of the Research Group LOGISIM, Universitat Autònoma de Barcelona. He has authored several scientific papers and research books. His current research interests include logistic systems, the development of decision support systems, causal modeling, and discrete event simulation.

Dr. Piera was nominated by the Society for Computer Simulation for the Outstanding Professional Contribution Award in 2013.

Antoni Guasch received the Ph.D. degree from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 1987.

He is an Associate Professor with the Department of Automatic Control, UPC, and the Head of Simulation and Industrial Optimization, inLab FIB, Barcelona. Since 1990, he has led over 40 industrial and research projects related to the modeling, simulation, and optimization processes for the nuclear, textile, transportation, automotive, water, pharmaceutical, and steel industries. His current research interests include modeling, simulation, and optimization of dynamic systems.