Deadlock-Free Scheduling and Control of Flexible Manufacturing Cells Using Automata Theory

Hamid Reza Golmakani, James K. Mills, *Member, IEEE*, and Beno Benhabib

***Abstract*—This paper presents a novel method for the scheduling and control of flexible manufacturing cells (FMCs). The approach employs automata, augmented by time labels proposed herein, for the modeling of machines, transportation devices, buffers, precedence constraints, and part routes. Ramadge–Wonham's supervisory-control theory is then used to synthesize a deadlock-free controller that is also capable of keeping track of time. For a given set of parts to be processed by the cell, A* search algorithm is subsequently employed using a proposed heuristic function. Three different production configurations are considered: Case 1) each part has a unique route; Case 2) parts may have multiple routes, but same devices in each route; and Case 3) parts may have multiple routes with different devices. The proposed approach yields optimal deadlock-free schedules for the first two cases. For Case 3, our simulations have yielded effective solutions but in practice, optimal deadlock-free schedules may not be obtainable without sacrificing computational time efficiency. One such nontime-efficient method is included in this paper.**

The proposed approach is illustrated through three typical manufacturing-cell simulation examples; the first adopted from a Petri-net-based scheduling paper, the second adopted from a mathematical-programming-based scheduling paper, and the third, a new example that deals with a more complex FMC scenario where parts have multiple routes for their production. These and other simulations clearly demonstrate the effectiveness of the proposed automata-based scheduling methodology.

***Index Terms*—Automata, deadlock-free scheduling, heuristic search, supervisory control.**

I. Introduction

SCHEDULING and control of flexible manufacturing cells (FMCs) is a complex problem. On the one hand, due to their flexibility in the production of multiple parts through multiple routes, there may exist several alternative schedules for the production of parts; on the other hand, the generated schedules must satisfy the controller's requirements in avoiding deadlocks, where at least two machines await circularly for each other to finish their respective operations, buffer overflows or underflows, etc. In this context, schedules that satisfy the control requirements and improve FMCs' performances would be highly desirable.

Manuscript received February 13, 2003; revised October 17, 2003, March 18, 2004, and July 27, 2004. This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada. The work of H. R. Golmakani was supported in part by the Ministry of Research, Science, and Technology (MRST) of Iran. This paper was recommended by Associate Editor M. Jeng.

The authors are with the Computer Integrated Manufacturing Laboratory, Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, ON M5S 3E5, Canada (e-mail: golmakni@mie.utoronto.ca).

Digital Object Identifier 10.1109/TSMCA.2005.851338

Mathematical programming is a typical traditional methodology for scheduling [1]. In [2], for example, job shop scheduling with a single production route for each part type was formulated as a 0/1 integer programming (0/1-IP) model with two simplifying assumptions: material-handling time is negligible and buffer space is infinite. Such impractical assumptions, however, may result in schedules that may not be implementable for FMCs due to potential deadlocks. In [3], an effort was made to incorporate material-handling times and limited buffer space into the model at the expense of increasing the numbers of decision variables and constraints. Clearly, incorporating FMC's routing flexibility would further increase the size of the 0/1-IP model. Despite some proposed methods for complexity reduction in solving the 0/1-IP model (e.g., [4]), the computational complexity and difficulties to formulate all FMC characteristics still remain as the two main restrictions in applying this approach in practice.

Dispatching-rule-based heuristics have also been widely developed and proposed for scheduling, e.g., [5] and [6]. However, no one dispatching rule has been found to be dominant [2]. Neural network [7], fuzzy logic [8], [9], simulation [10], and decision-tree constructed based on a learning procedure [11], [12] have been proposed to determine the best dispatching rule out of a predefined set of rules. It has been shown that, for a given performance criterion and a cell configuration, changing the rules dynamically over the scheduling horizon may improve the FMC's performance [13]–[15]. Nevertheless, dispatching rules, by themselves, only resolve the question of choice by considering some part/machine attributes and inherently lack the capability to determine optimal deadlock-free choices.

Local search methods such as simulated annealing, tabu search, and genetic algorithm attempt to find a schedule better than the current one through a search in the neighborhood of the current schedule. In addition to the fact that they do not guarantee finding the optimal schedule, their effectiveness significantly depends on the initial schedule, the design of the neighborhood, and the movement direction [2].

Branch-and-bound methods can obtain optimal solutions to scheduling problems. However, enumerating large number of nodes is their main disadvantage. Other search methods such as filtered-beam search [16] and A* search algorithm [17]–[20], which are based on branch-and-bound methods, attempt to generate and evaluate fewer number of nodes by branching only the most promising nodes at each stage of the search.

Since none of the search methods mentioned above can essentially take into account the problem of deadlocks, a deadlock-detection phase is required to be incorporated into the search process in order to generate deadlock-free schedules.

This must be done by precisely representing the FMC's discrete-event dynamic behavior to prevent encountering deadlock states. Petri nets (PNs) and automata are two commonly accepted techniques that can explicitly represent FMC characteristics [21]–[23].

For an FMC modeled by PNs configured to process a given batch of parts, an optimal schedule can be obtained by generating the reachability tree and finding the optimal path from the initial marking to the final marking based on a given measure of performance [24]–[26]. When constructing the reachability tree, deadlock states are designated and the paths leading to those states are terminated to prevent the system from encountering them [27]–[29]. For example, in [20] a hybrid search, based on an A^* search algorithm, was applied to determine a deadlock-free schedule with respect to the makespan criterion, assuming that each part has only one route to be produced. For PN methods, deadlock recognition is carried out as the schedule is generated and thus, it has to be repeated for every new batch of parts.

In contrast to PNs, automata theory, when used with the supervisory-control theory developed by Ramadge and Wonham (R–W theory) [22], would directly yield deadlock-free FMC supervisors "by construction" [13], [30]–[32]. Such supervisors determine formally the set of states FMC could reach as well as the set of events allowed to occur at those states. Therefore, the deadlock detection phase is completed once the supervisor is generated. The supervisor can, then, explicitly represent all possible deadlock-free sequences of events (i.e., schedules).

Since R–W supervisors were originally designed to emphasize the state and event sequences FMC may undergo without considering timing issues, they need to be time augmented for scheduling purposes. Thus, in this paper, we propose a novel method to augment automata with timing requirements for FMC scheduling purposes. We employ R–W theory to construct a deadlock-free search space to be utilized by A^* search algorithm to obtain deadlock-free schedules.

II. Automata-Based Modeling Controllers Synthesis

A. Time-Augmented Automata

A finite-state automaton (FA) is often described by a four-tuple $\mathrm{FA} = \{\Sigma, Q, q_0, \delta(q,\sigma)\}$, [22], [33]: Σ is a finite set of events; Q is a finite set of states; q_0 is the initial state; and δ is the transition function mapping $Q \times \Sigma$ to Q. The function $\delta(q,\sigma)$ defines the next state for each state–event symbol pair ($q \in Q$, $\sigma \in \Sigma$). We denote $\delta(q,\sigma)!$, if $\delta(q,\sigma)$ is defined, namely, σ is accepted at state q.

In order to use FA as a basic modeling tool for the scheduling and control of an FMC, its definition needs to be augmented with time. An integer global clock T and a variable C for each individual automaton, for example C_{A} for automaton A, are proposed herein. The variable, C_{A}, is reset to T every time an event changes the current state of the automaton. The value of C_{A} provides information about the occurrence time of the last event. A label that is a function of C_{A}, T, and the part-processing time is associated to each event. The value of this label, corresponding to a given event at a given state, represents the earliest time that it may occur. At a state where multiple future events are possible, the comparison of the values of the events' labels would indicate the event that is most likely to occur first.

The above proposed time-augmented automata (AFA) is formally defined as a six-tuple $\mathrm{AFA} = \{\Sigma, Q, q_0, \delta(q,\sigma), \Delta, \omega\}$, where Δ is a finite set of labels and ω is a one-to-one function that maps each event $\sigma \in \Sigma$ to its associated label, ω: $(\sigma \in \Sigma) \rightarrow \Delta$.

B. Basic Logic Operations on AFA

1) The Shuffle Operation: The shuffle operation on two AFAs, AFA_1 and AFA_2, yields an AFA consisting of all possible interleaving strings, which represents the coordinated behavior of both AFA. Based on [32], the shuffle operation $\otimes$ on two AFAs, AFA_1 and AFA_2, is defined herein as

$$\begin{aligned}&\mathrm{Shuffle}(\mathrm{AFA}_1, \mathrm{AFA}_2)\\ &\quad = \mathrm{AFA}_1 \otimes \mathrm{AFA}_2\\ &\quad = \left\{ \bigcup_{n=1,2} (\Sigma)_n, Q, q_0, \delta^*(q,\sigma^*), \quad \bigcup_{n=1,2} (\Delta)_n, \omega^* \right\} \end{aligned} \tag{1}$$

where $\bigcup_{n=1,2}(\Sigma)_n$ is the finite set of events defined as $\bigcup_{n=1,2}(\Sigma)_n = \sum_1 \cup \sum_2$; and Q is the Cartesian product of Q_1 and Q_2, $Q_1 \times Q_2$, which defines all possible global states of the resulted AFA. A global state $q \in Q$ is defined as a pair: ($q_{i_1} \in Q_1$, $q_{j_2} \in Q_2$), where q_{i_1} refers to State i of AFA_1 and q_{j_2} refers to State j of AFA_2; q_0 is the initial global state, $q_0 = (q_{0_1}, q_{0_2}) \in Q$; σ^* is an event: $\sigma^* \in \bigcup_{n=1,2}(\Sigma)_n$; and δ^* is the partial transition function, δ^*: $Q \times \bigcup_{n=1,2}(\Sigma)_n \rightarrow Q$ such that

$$\delta^*\left((q_{i_1}, q_{j_2}), \sigma^*\right) = \begin{cases} \delta_1(q_{i_1},\sigma_1)\, x\, \{q_{j_2}\}, & \text{if } \delta_1(q_{i_1},\sigma_1)! \wedge \delta_2(q_{j_2},\sigma_2) \\ \{q_{i_1}\}\, x \delta_2(q_{j_2},\sigma_2), & \text{if } \delta_2(q_{j_2},\sigma_2)! \wedge \delta_1(q_{i_1},\sigma_1) \\ \delta_1(q_{i_1},\sigma_1)\, x \delta_2(q_{j_2},\sigma_2), & \text{if } \delta_1(q_{i_1},\sigma_1)! \wedge \delta_2(q_{j_2},\sigma_2) \\ \varnothing, & \text{otherwise} \end{cases}. \tag{2}$$

Above, $\varnothing$ represents the empty set, i.e., no transition can occur from State (q_{i_1}, q_{j_2}); $\bigcup_{n=1,2}(\Delta)_n$ is the finite set of labels defined as $\bigcup_{n=1,2}(\Delta)_n = \Delta_1 \cup \Delta_2$; and ω^* is the function that maps each event $\sigma^* \in \bigcup_{n=1,2}(\Sigma)_n$ to its associated label such that

$$\omega^*(\sigma^*) = \begin{cases} \ell_1 \in \Delta_1, & \text{if } \sigma^* \in \sum_1 \\ \ell_2 \in \Delta_2, & \text{if } \sigma^* \in \sum_2 \end{cases}. \tag{3}$$

2) The Meet Operation: The meet operation on two AFAs, AFA_1 and AFA_2, yields an AFA that represents the synchronized behavior of both. However, if the AFA have no common event, $\cap_{n=1,2}(\Sigma)_n = \varnothing$, there is no synchronization to be

performed and the resulting AFA is empty, i.e., an AFA with one state and zero transition. Based on [32], the meet operation $\oplus$ on two AFAs is defined herein as

$$\begin{aligned}&\mathrm{Meet}(\mathrm{AFA}_1, \mathrm{AFA}_2)\\&\quad = \mathrm{AFA}_1 \oplus \mathrm{AFA}_2\\&\quad = \{\cap_{n=1,2}(\Sigma)_n, Q, q_0, \delta^*(q,\sigma^*), \quad \cap_{n=1,2}(\Delta)_n, \omega^*\} \quad (4)\end{aligned}$$

where Q, q_0, δ^*, σ^*, and q are defined same as above, whereas $\cap_{n=1,2}(\Sigma)_n = \sum_1 \cap \sum_2$; and δ^* is defined as

$$\delta^*\left((q_{i_1}, q_{j_2}), \sigma^*\right) = \begin{cases} \delta_1\left(q_{i_1}, \sigma^*\right) x \delta_2\left(q_{j_2}, \sigma^*\right), & \text{if } \delta_1\left(q_{i_1}, \sigma^*\right)! \wedge \delta_2\left(q_{j_2}, \sigma^*\right)! \\ \varnothing, & \text{otherwise} \end{cases}. \quad (5)$$

Above, $\cap_{n=1,2}(\Delta)_n$ is the finite set of labels defined as $\cap_{n=1,2}(\Delta)_n = \Delta_1 \cap \Delta_2$; and ω^* is the function mapping each event $\sigma^* \in \cap_{n=1,2}(\Sigma)_n$ to its associated label, $\omega^*(\sigma^*) = \ell \in \cap_{n=1,2}(\Delta)_n$.

C. AFA for Supervisor Synthesis

In R–W control theory, the synthesized supervisor represents the unrestricted behavior of the plant, where events are divided into two disjoint sets of controllable and uncontrollable events, respectively. This supervisor enables or disables controllable events such that the language (sequence of events) generated by the closed-loop-controlled plant satisfies the plant specifications, also defined by finite automata. One notes, however, that although an R–W supervisor represents the maximally deadlock-free behavior of the FMC, this supervisor could include states with multiple controllable events as their outputs, respectively. Namely, the question of choice may arise at such states necessitating the use of a decision-making agent. These decisions, consequently, would determine the performance of the FMC.

1) FMC Modeling: In order to synthesize an FMC supervisor, individual cell devices (i.e., machines, robots, etc.), part routes, and specifications (i.e., buffer capacity, machine repair priority, etc.) need to be modeled individually. A cell device model represents the behavior of a workcell machine. The model, typically consists of three states, Idle (I), Working (W), and broken-Down (D), and four types of events, namely, Type α for operation started, Type β for operation finished, Type λ for machine broken-down, and Type μ for machine repaired. Time labels are associated to events based on the processing times. In Fig. 1, for example, at State I, two events may occur, $(\alpha, M1, \mathbf{A}, 1)$ and $(\alpha, M1, \mathbf{B}, 2)$. The corresponding time labels indicate that the earliest time one of these events may occur is 0. At State $\mathrm{W_A}$, on the other hand, the time label $(\beta, M1, \mathbf{A}, 1)$: $C_1 + 25 - T$ indicates the expected time the event $(\beta, M1, \mathbf{A}, 1)$ may occur, i.e., the end of operation on Part $\mathbf{A}$.

A basic part-route model, denoted by $\mathbf{BPM_X}$ represents the full production sequence for Part X. It is a sequence of states and events in a treelike form, where each branch represents a specific route. As an example, Fig. 2 illustrates the $\mathbf{BPM_A}$ for Part $\mathbf{A}$ with two alternative routes in an FMC having two machines $M1$ and $M2$ and a robot $R1$ (for simplicity of illustration, the down state of the machines and the robot were not considered when constructing this model).

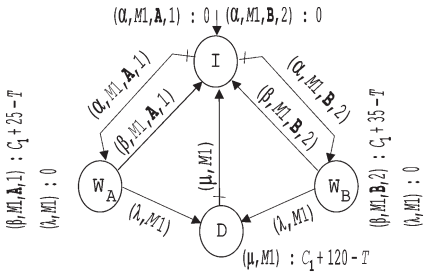

Fig. 1. Machine model.

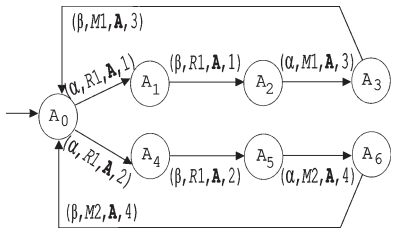

Fig. 2. Basic part-route model for Part $\mathbf{A}$, $\mathbf{BPM_A}$.

When synthesizing a supervisor, one has to consider the possibility of having multiple parts of the same type in the FMC concurrently. In order to cope with such cases, advanced part-route models $\mathbf{APM_X}$ must be constructed by performing $(n-1)$ shuffle operations on $\mathbf{BPM_X}$

$$\mathbf{APM_X} = \mathbf{BPM_{X_1}} \otimes \mathbf{BPM_{X_2}} \otimes \cdots \otimes \mathbf{BPM_{X_n}} \quad (6)$$

where n is the maximum number of parts of type X that the FMC can concurrently process. However, the resulting $\mathbf{APM_X}$ may have deadlock states, in which some or all of the parts that are currently being processed cannot proceed to completion. Such deadlock states are "trimmed" by deleting the transitions leading to or coming from them. The function that determines the deadlock states and erases them is represented by a Safe operation [31] and the resulting model, which can trace the history of the specific part type being concurrently produced by the workcell, is denoted by $\mathbf{APM_X^s}$

$$\mathbf{APM_X^s} = \mathrm{safe}\,(\mathbf{APM_X}). \quad (7)$$

Manufacturing specifications for an FMC may be interpreted as the desired behavior of the workcell that must be imposed by the supervisor. Buffer overflow or underflow and machine repair priorities are some typical examples. For instance, let us consider a two-machine FMC, Machines $M1$ and $M2$, with only one part type $\mathbf{A}$ and a buffer of capacity 1. Part $\mathbf{A}$ is first processed by $M1$, then released to the buffer and finally processed by $M2$. The buffer model shown in Fig. 3 consists

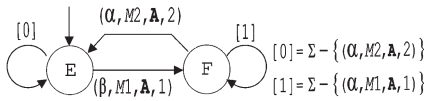

Fig. 3. Typical specification preventing buffer underflow and overflow.

of two states, Empty (E) and Full (F). The event $(\alpha, M2, \mathbf{A}, 2)$ is disabled at State E in order to prevent buffer underflow and the event $(\alpha, M1, \mathbf{A}, 1)$ is disabled at Sate F in order to prevent buffer overflow, respectively.

2) Supervisor Synthesis: Once all the individual models have been constructed, the FMC supervisor is synthesized by performing the following sequence of logic operations

$$\begin{aligned}
&\mathbf{CELL} = \mathbf{M}_1 \otimes \mathbf{M}_2 \otimes \ldots \otimes \mathbf{M}_n \\
&\mathbf{PARTS} = \mathbf{APM}^{\mathrm{s}}_{\mathbf{A}} \otimes \mathbf{APM}^{\mathrm{s}}_{\mathbf{B}} \otimes \ldots \otimes \mathbf{APM}^{\mathrm{s}}_{\mathbf{K}} \\
&\mathbf{SUPER} = \mathbf{CELL} \oplus \mathbf{PARTS} \\
&\quad For\ p = 1\ to\ number\ of\ specifications \\
&\qquad \mathbf{SUPER} = \mathbf{SUPER} \oplus \mathbf{SPEC}_p \\
&\quad End\ \ for.
\end{aligned} \tag{8}$$

Above, **CELL** is the unconstrained behavior of FMC, **PARTS** represents all possible sequences in which parts $\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots, \mathbf{K}$ can be concurrently processed, $\mathbf{SPEC}_p$ is the pth specification, and **SUPER** is an AFA that restricts the unconstrained behavior of FMC to only those that can satisfy **PARTS** and all $\mathbf{SPEC}_p$ simultaneously. The time labels in **SUPER** enable us to keep track of time and determine the most likely event to occur at each state of the model.

III. FMC Scheduling Using AFA Supervisor

Given an FMC and a set of parts to be produced, the scheduling problem is to find a sequence of decisions for manufacturing the parts that optimizes a measure of performance. This can be considered as a search of a node-decision tree where at each decision-making instance, represented by a node, multiple controllable events are available to the decision maker to choose from. The initial node represents the start of the scheduling problem and the goal node is the node where all the parts have been manufactured. Any path that starts at the initial node and ends at the final node represents a solution to the scheduling problem.

In this section, an efficient methodology is proposed to generate a partial node-decision tree required to search for optimal deadlock-free decisions with respect to makespan as the performance criterion. This is achieved by using jointly the AFA supervisor and an A* search algorithm. The direction of node generation is determined by the "promise" of the node, which is estimated by a heuristic function. The most promising direction is evaluated first. Once a direction is selected, the AFA supervisory model is used to determine the next decision node, the time it would be reached, and the possible choices available at that node. When the goal node is reached, tracing back from the goal node to the initial node yields the schedule.

A. Node Generation

The state of the given batch, together with the state of the supervisor **SUPER** determines the stage (node) of the FMC scheduling problem. We denote a node i by a pair $N_i = (L_j, S_k)$, where L_j denotes the state of the given batch ($j = 0$ to the number of possible states of the batch) and S_k denotes the state of the supervisor ($k = 0$ to the number of states in the **SUPER**). For example, for an FMC capable of producing three part types **A**, **B**, and **C**, the string $L_0 = [2\ 3\ 4]$ depicts the initial state of the batch, namely, at State 0 the batch consists of 2, 3, and 4 units of **A**, **B**, and **C**, respectively. The state of the batch changes once the FMC starts to operate (e.g., $L_1 = [1\ 3\ 4]$, $L_1 = [2\ 2\ 4]$, or $L_1 = [2\ 3\ 3]$) depending on the decision made by the scheduler regarding which part type to process first.

With the above nomenclature, the initial node of the scheduling problem is denoted by $N_0 = (L_0, S_0)$: the batch and the supervisor are at their initial states, where S_0 indicates that all the FMC's devices are idle. The goal node is denoted by $N_{\mathrm{g}} = (L_{\mathrm{g}}, S_0)$, where $L_{\mathrm{g}} = [0\ 0\ 0]$. The scheduling problem with respect to makespan is, thus, defined as finding a sequence of nodes that minimizes the time needed to reach N_{g} from N_0.

At a given node, the α-type events (i.e., start of operations) that are allowed to occur by the controller represent the decisions available to the scheduler. For α-type events associated with the first operation of a part, the existence of the part in the batch must be also examined. One can recall that the execution of the controller only generates deadlock-free nodes. The clock structure and labels, defined in the **SUPER**, keep track of time, determine the time β-type events (i.e., completion of operations) may happen, and finally, compute the time taken to reach successive decision nodes.

B. Heuristic Function

The cost function $\mathcal{F}(N)$, computed for a node N, guides the direction of the search [34]. When using makespan as the performance criterion, $\mathcal{F}(N)$ is defined as the estimated completion time of all the parts via an optimal path, starting at the initial node and ending at the goal node while passing through the node N: $\mathcal{F}(N) = \mathcal{G}(N) + \mathcal{H}(N)$. The term $\mathcal{G}(N)$, which is the time required to reach node N from the initial node, is computed based on the execution of the search over the AFA supervisor. The term $\mathcal{H}(N)$, referred to as the heuristic function, is an estimate of the time required to reach the goal node from the current node N.

The minimum time to reach the goal node $\mathcal{H}^*(N)$ is not known at node N. With the A* search algorithm, any heuristic function that satisfies $\mathcal{H}(N) \leq \mathcal{H}^*(N)$, for all N, guarantees obtaining the optimal solution [34]. Setting $\mathcal{H}(N)$ equal to zero for all N, as a possible inefficient option, leads the search to obtain the optimal solution, but at a cost of extensive node generation and thus, computation complexity. The closer the value derived from the heuristic function is to $\mathcal{H}^*(N)$, the fewer

the number of node expansions. Hence, it is desirable to employ a heuristic function that yields values less than or equal (as close as possible) to $\mathcal{H}^*$ for all nodes and that can be computed by the information encoded by the node.

We employ the following heuristic for the makespan.

Step 1) At a given node N, consider all the parts that are either waiting in the batch or those that are being processed by the FMC.

Step 2) Calculate the sum of the operation times of those remaining operations for the parts specified in Step 1 on each device of the FMC (machine, robot, etc.).
   a) When each part has a unique route, the processing time on each machine is considered (Case 1).
   b) When some parts have multiple routes, but the required devices in each alternative route are the same, the minimum processing time on each machine is considered for the machine (Case 2).
   c) When some parts have multiple routes, but with a different set of required devices, the average of times taken by each machine is considered for the machine (Case 3).

Step 3) Let $U_i(N)$ be the total time required for Device i at node N. $\mathcal{H}(N)$ is then equal to $\max_i\{U_i(N)\}$.

C. Search Algorithm

The following algorithm is proposed for deadlock-free FMC scheduling. It employs a A* search algorithm that utilizes the AFA supervisor **SUPER** for node generation. The algorithm starts with the initial node. At each step of the search, the most promising node that has been generated so far is selected. The promise of a node, i.e., the direction of node expansion, is determined by the proposed heuristic function. Then, the node is expanded using **SUPER**. If one of the proceeding nodes is the goal node, the algorithm stops and the solution is determined by tracing back from the goal node to the initial node; otherwise, the nodes are added to a list for further exploration. In order to distinguish between nodes that have been explored and those that have been generated but not yet explored, they are placed into two separate lists OPEN and CLOSED, respectively.

1) Place the initial node N_0 into OPEN. Set the system clock T to zero.
2) Retrieve from OPEN the node N for which F has the minimum value and place it into CLOSED. In the case of multiple choices, select one arbitrarily.
   a) If node N is the goal node, stop. The solution is obtained by tracing back the pointers from current N to N_0.
   b) Otherwise, using **SUPER**, consider all possible events that are allowed to occur at node N and generate their proceeding node(s), N', using **SUPER**. Assign pointers back to N.
3) For each N' compute $F(N')$ and $T_{N'}$.
   a) If N' was neither in OPEN nor in CLOSED, add it to OPEN. Assign the newly computed $F(N')$ and $T_{N'}$ to it.
   b) Otherwise, compare the newly computed $F(N')$ with the previously assigned value. If the old value is lower, discard the new one. If the new value is lower, substitute it for the old one. If the matching node N' was in CLOSED, move it back to OPEN.

Return to Step 2).

In Step 2b), all the controllable events enabled at the current state of **SUPER** are considered as possible expansions. Having selected each event one at a time, the next state of the system is determined using **SUPER** (state of the batch also changes if the event is associated to the first operation of the part). If the new state is a decision state (a state with more than one controllable event), it represents a new node. Otherwise, at the new state, the values of all labels are calculated. The event with the minimum value is selected as if it were to happen once the workcell reaches the new state. This is repeated until a new decision state, a new node, is reached.

D. Optimality of Solutions

The A* search algorithm guarantees obtaining the optimal solution if $\mathcal{H}(N) \leq \mathcal{H}^*(N)$, for all nodes N [34]. $\mathcal{H}(N)$ is an estimation of the time to finish all the remaining operations of all the parts existing at node N and $\mathcal{H}^*(N)$ is the actual minimum one. Clearly, $\mathcal{H}^*(N)$ is greater than or equal to $\max_i\{U_i(N)\}$, where $U_i(N)$ is the total remaining workload on Machine i. Namely, $\mathcal{H}^*(N)$ cannot be less than the time the busiest device requires to finish its assigned operations. This fact allowed us to develop a heuristic function that calculates the remaining workload on each device to obtain the maximum workload as an estimation for $\mathcal{H}^*(N)$. One can note that the set of remaining parts at each node is determined by the state of the batch together with the state of the supervisor.

In Case 1 of the heuristic, the calculation of the remaining workload on each machine is straightforward since each part type in this specific case has a unique route and therefore, the maximum workload will not exceed $\mathcal{H}^*(N)$. In Cases 2 and 3 where parts have multiple routes, the remaining workload on each machine depends on how parts are assigned to their possible routes. Clearly, the assignment that minimizes the maximum workload among all devices yields a value for the heuristic function less than or equal to $\mathcal{H}^*(N)$. This part-to-route assignment problem can be formulated via an IP model

$$\min Z = \max_k \left\{ \sum_{i=1}^{I} \sum_{j=1}^{J_i} T_{ijk} X_{ij} \right\} \qquad k = 1, 2, \ldots, K$$

$$\sum_{j=1}^{J_i} X_{ij} = D_i \qquad i = 1, 2, \ldots, I$$

$$X_{ij} \geq 0 \quad \text{and integer.} \tag{9}$$

Above, the decision variable X_{ij} determines the number of parts of Type i allocated to Route j, I is the total part types and D_i $(i = 1, 2, \ldots, I)$ is the number of units of part Type i at node N, J_i is the number of possible routes for the production of part Type i, K is the number of devices within FMC, and T_{ijk} is the operation time of part Type i using Route j on Device k $(k = 1, 2, \ldots, K)$.

The objective function in (9) is linearized as follows:

$$\min Z = y$$

$$\text{Subject to}: \sum_{j=1}^{J_i} X_{ij} = D_i \qquad i = 1, 2, \ldots, I$$

$$\sum_{i=1}^{I}\sum_{j=1}^{J_i} T_{ijk} X_{ij} \le y \qquad k = 1, 2, \ldots, K$$

$$y \ge 0, \quad X_{ij} \ge 0 \text{ and integer.} \tag{10}$$

The solution of the above IP model determines the optimal assignments of parts to their alternative routes that minimize the maximum potential workload among all devices. The value of Z is clearly less than or equal to $\mathcal{H}^*(N)$ and can guarantee the A* search obtaining the optimal solution for the original scheduling problem in Cases 2 and 3, where parts have multiple routes. However, it requires solving the IP model at all nodes, which obviously increases the computation complexity. Instead of solving the IP model at each node, the proposed heuristic function estimates the maximum workload for both cases.

In Case 2, where parts have multiple routes and the required devices in each alternative route are the same, the heuristic assumes that whenever a part visits a device, no matter which route has been taken by the part, it is processed in the minimum possible time the part may spend at the current device. This can be thought of as if the part has only one route but with its associated minimum processing times. With this optimistic assumption, the maximum workload derived by the heuristic is clearly less than the workload derived by the IP model Z due to the fact that the least processing times are assumed in the calculations. Subsequently, this workload is less than the $\mathcal{H}^*(N)$ and thus, the A* search algorithm yields the optimal solution.

In Case 3, where parts have multiple routes and the required devices are different, the computation of the maximum workload is carried out assuming that parts are equally assigned to their potential routes. Thus, the estimation of maximum workload obtained from the heuristic function may be greater than the one that is derived from the IP model Z. However, one may note that a value greater than Z does not necessarily mean that it is greater than $\mathcal{H}^*(N)$. Since $\mathcal{H}^*(N)$ is the optimal completion time of all remaining operations at node N, this time, for most realistic cases, is much more than Z, namely, the minimum of maximum potential workloads. This is due to the fact that the priority requirements among the operations and the FMC's limited buffers cause delays in operations. Therefore, since there might be some instances where the value calculated by the heuristic exceeds $\mathcal{H}^*(N)$, the search might not be able to generate the optimal solution, but it is most likely to find practically acceptable solutions in this specific case.

IV. Simulation Examples

In order to illustrate that the proposed approach can perform well under a variety of FMC configurations, several simulation examples were tested on a Dell Dimension 4300 computer, 1.5-GHz Pentium 4 with 128 MB random access memory. Three of these examples are presented in this Section. The first two examples, adopted from [20] and [3], a PN- and mathematical-programming-based scheduling approach, respectively, deal with Case 1, where each part has a single route. Example 3, where parts have multiple routes, is presented to illustrate the effectiveness of the proposed approach under Case 2, Case 3, and all combinations of the three Cases.

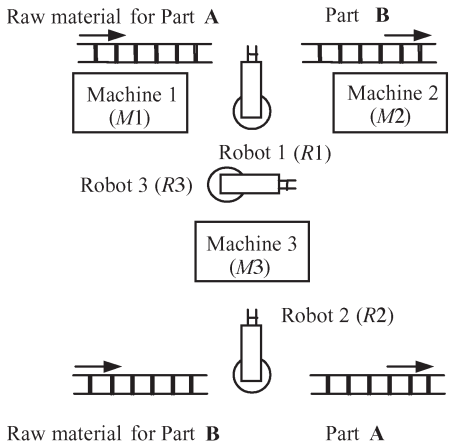

Fig. 4. Layout of the workcell.

TABLE I
Sequence of Operations for Parts **A** and **B**

Part Type	Operation Number	Device	Processing Time	Description
A	1	*R*1	0	*R*1 loads *M*1.
	2	*M*1	5	*M*1 performs first operation.
	3	*R*3	0	*R*3 unloads *M*1 and loads *M*2.
	4	*M*2	8	*M*2 performs second operation.
	5	*R*3	0	*R*3 unloads *M*2 and loads *M*3.
	6	*M*3	2	*M*3 performs third operation.
	7	*R*2	0	*R*2 unloads *M*3.
B	a	*R*2	0	*R*2 loads *M*3.
	b	*M*3	7	*M*3 performs first operation.
	c	*R*3	0	*R*3 unloads *M*3 and loads *M*1.
	d	*M*1	3	*M*1 performs second operation.
	e	*R*3	0	*R*3 unloads *M*1 and loads *M*2.
	f	*M*2	9	*M*2 performs third operation.
	g	*R*1	0	*R*1 unloads *M*2.

A. Example 1

Let us consider a three-machine and three-robot workcell, adopted from [20], with two types of parts, **A** and **B**, each having one route. The layout of the workcell is shown in Fig. 4. The sequence and processing times for each operation on each machine corresponding to each part type are given in Table I. As in [20], loading and unloading times are neglected.

Based on the modeling method described in Section II, the individual device models were derived first (for the sake of simplicity of illustration, down states were not considered in this example), as seen in Fig. 5, where $\mathbf{CELL} = \mathbf{M}_1 \otimes \mathbf{M}_2 \otimes \mathbf{M}_3 \otimes \mathbf{R}_1 \otimes \mathbf{R}_2 \otimes \mathbf{R}_3$. Subsequently, the basic part-route models for Parts **A** and **B**, $\mathbf{BPM_A}$ and $\mathbf{BPM_B}$, were derived, as seen in Fig. 6. Since the workcell can hold a maximum of six parts of each type at any time, the automaton $\mathbf{APM}_{\mathbf{A}}^{\mathbf{s}} = \text{safe}(\mathbf{BPM_A} \otimes \mathbf{BPM_A} \otimes \mathbf{BPM_A} \otimes \mathbf{BPM_A} \otimes \mathbf{BPM_A} \otimes$

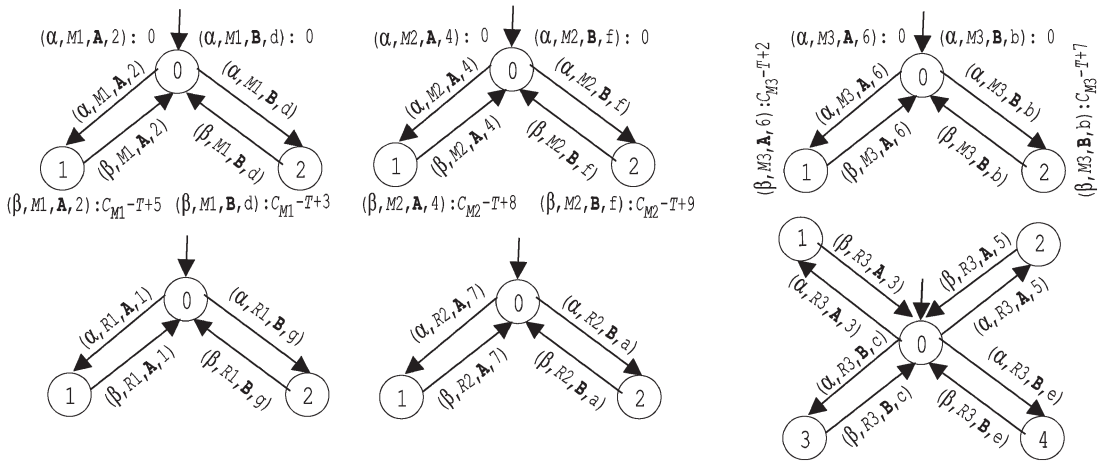

Fig. 5. Top row: Models for machine $M1$, $M2$, and $M3$. Bottom row: Models for robot $R1$, $R2$, and $R3$. For simplicity, labels in robot models are not shown.

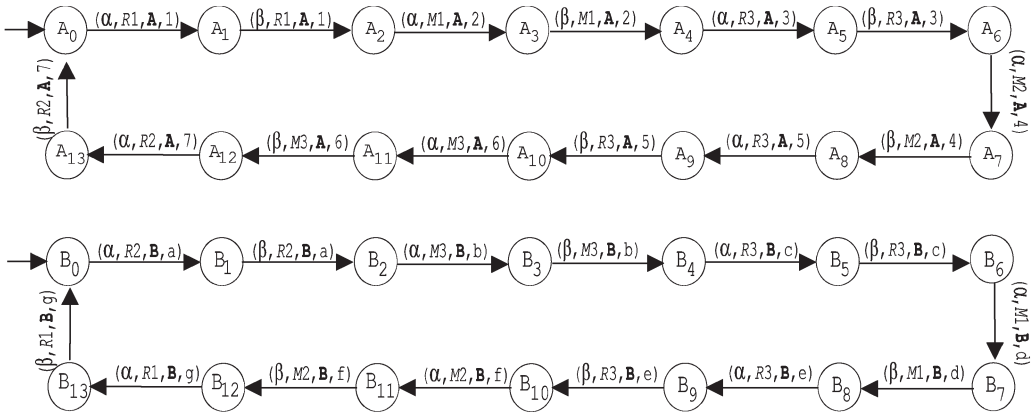

Fig. 6. Basic part-route models for Parts $\mathbf{A}$ and $\mathbf{B}$.

TABLE II
SEARCH RESULTS FOR BATCHES

Batch Sizes	Makespan	Number of Node Expansions		Computation Time (second)	
		PN-based [20]	Proposed Method	PN-based [20]	Proposed Method
1	22	13	7	-	1
2	39	25	17	-	2.6
5	90	101	47	-	8.1
10	175	230	97	-	19

$\mathbf{BPM_A}$). $\mathbf{APM_B^s}$ is modeled similarly. Thus, $\mathbf{PARTS} = \mathbf{APM_A^s} \otimes \mathbf{APM_B^s}$. Three manufacturing constraints $\mathbf{SPEC}_1$, $\mathbf{SPEC}_2$, and $\mathbf{SPEC}_3$ were considered next. They ensure that a machine can start its operations only when the corresponding robot has loaded the machine. They also ensure that a robot cannot load a machine with a new part unless the previous part has been unloaded. Finally, the automaton $\mathbf{SUPER}$, representing the supervisor, was obtained using (8). It represents the deadlock-free behavior of the workcell, while having 656 states and 1638 transitions. The time spent by the Synthesiser Module to generate this automaton was 56 s.

As in [20], batch sizes of 1, 2, 5, and 10 (for each part type), respectively, were considered. Table II presents the makespan, the number of nodes generated, and the computation time required finding the solution for each batch size. One should note that, in [20], computation times were not provided.

As an example, let us consider the batch size of 1, i.e., one unit of each part types $\mathbf{A}$ and $\mathbf{B}$. Fig. 7 shows the node-decision graph generated by performing the search algorithm. The algorithm starts at node 1, the initial node. This node is denoted by $N_1 = (0, [1, 1])$ representing the state of the $\mathbf{SUPER}$, 0, and the state of the batch, [1,1], i.e., one unit of each part type. Time T is zero. $\mathcal{G}(1)$, the time spent to reach the node 1 from the initial node, is clearly zero. According to the heuristic function, in order to find the value of $\mathcal{H}(1)$, the workload on each device is calculated first

$$[U_{M1}\ U_{M2}\ U_{M3}\ U_{R1}\ U_{R2}\ U_{R3}] = [1\ 1]\begin{bmatrix} 5 & 8 & 2 & 0 & 0 & 0 \\ 3 & 9 & 7 & 0 & 0 & 0 \end{bmatrix}$$
$$= [8\ 17\ 9\ 0\ 0\ 0].$$

Thus, $\mathcal{H}(1) = \max\ (8, 17, 9, 0, 0, 0) = 17$ and $\mathcal{F}(1) = \mathcal{G}(1) + \mathcal{H}(1) = 17$. According to the algorithm, this node is placed into OPEN, Step 1). Among the nodes in OPEN, a node with the minimum value of $\mathcal{F}$ is removed and placed into CLOSED. Node 1 is removed, Step 2). Case 2a) is not satisfied since it is not the goal node, $N_g = (0, [0, 0])$. According to Step 2b), controllable events allowed at the current state of $\mathbf{SUPER}$ are considered, $(\alpha, R1, \mathbf{A}, 1)$ and $(\alpha, R2, \mathbf{B}, \mathrm{a})$. For each possible decision, the next decision node is determined using $\mathbf{SUPER}$, nodes 2 and 3. Then, for each newly

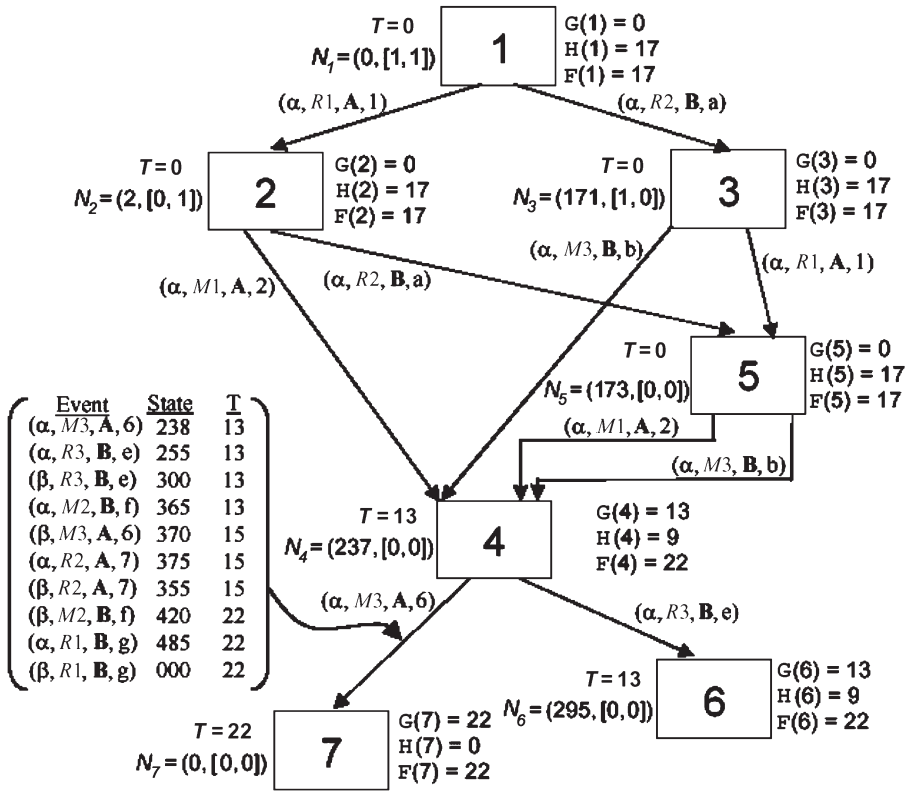

Fig. 7. Nodes generated by the search algorithm for batch size of 1.

TABLE III
OPTIMAL SCHEDULE FOR BATCH OF 1

Row	Event	Time to Occur		Row	Event	Time to Occur
1	$(\alpha, R1, \mathbf{A}, 1)$	0		15	$(\beta, M1, \mathbf{B}, \mathrm{d})$	10
2	$(\beta, R1, \mathbf{A}, 1)$	0		16	$(\beta, M2, \mathbf{A}, 4)$	13
3	$(\alpha, M1, \mathbf{A}, 2)$	0		17	$(\alpha, R3, \mathbf{A}, 5)$	13
4	$(\alpha, R2, \mathbf{B}, \mathrm{a})$	0		18	$(\beta, R3, \mathbf{A}, 5)$	13
5	$(\beta, R2, \mathbf{B}, \mathrm{a})$	0		19	$(\alpha, M3, \mathbf{A}, 6)$	13
6	$(\alpha, M3, \mathbf{B}, \mathrm{b})$	0		20	$(\alpha, R3, \mathbf{B}, \mathrm{e})$	13
7	$(\beta, M1, \mathbf{A}, 2)$	5		21	$(\beta, R3, \mathbf{B}, \mathrm{e})$	13
8	$(\alpha, R3, \mathbf{A}, 3)$	5		22	$(\alpha, M2, \mathbf{B}, \mathrm{f})$	13
9	$(\beta, R3, \mathbf{A}, 3)$	5		23	$(\beta, M3, \mathbf{A}, 6)$	15
10	$(\alpha, M2, \mathbf{A}, 4)$	5		24	$(\alpha, R2, \mathbf{A}, 7)$	15
11	$(\beta, M3, \mathbf{B}, \mathrm{b})$	7		25	$(\beta, R2, \mathbf{A}, 7)$	15
12	$(\alpha, R3, \mathbf{B}, \mathrm{c})$	7		26	$(\beta, M2, \mathbf{B}, \mathrm{f})$	22
13	$(\beta, R3, \mathbf{B}, \mathrm{c})$	7		27	$(\alpha, R1, \mathbf{B}, \mathrm{g})$	22
14	$(\alpha, M1, \mathbf{B}, \mathrm{d})$	7		28	$(\beta, R1, \mathbf{B}, \mathrm{g})$	22

generated nodes, the $\mathcal{F}$ values, $\mathcal{F}(2)$ and $\mathcal{F}(3)$, are computed, Step 3). Since neither node 2 nor 3 are in OPEN, they are added to OPEN, Step 3a). The process is repeated by removing the node with the minimum value of $\mathcal{F}$, node 2, from OPEN. Expansion of node 2 generates nodes 4 and 5 and thus, $\mathcal{F}(4) = 22$ and $\mathcal{F}(5) = 17$. Again, node 3 is evaluated, which leads to the same nodes, 4 and 5. At this point, the node with the minimum value of $\mathcal{F}$ is node 5. The evaluation of this node leads to node 4. Then, node 4 is removed for evaluation. Two nodes, 6 and 7 are generated. Since node 7 is the goal node, the search process stops and the optimal schedule is obtained by tracing back from State 7, to 4, to 2, and finally to 1. The optimal schedule for batch size of 1 is given in Table III.

As stated above, the major role of the supervisor in the search process is to determine the deadlock-free sequence of events between each pair of nodes as well as the time required to reach a node from the others. For example, at node 4, the selection of the controllable Event $(\alpha, M3, \mathbf{A}, 6)$ is allowed by the supervisor. One may note that α-type events, which are controllable events, can be immediately scheduled when the supervisor allows them. Selection of this event by the search algorithm leads the workcell to reach State 238. At this state, Event $(\alpha, R3, \mathbf{B}, \mathrm{e})$, which also is a controllable event, is the only choice. Selection of this event subsequently leads the workcell to State 255, where only the uncontrollable Event $(\beta, R3, \mathbf{B}, \mathrm{e})$ may happen. Here, there is no decision to be made. One may note, however, that the search determines the time β-type events happen, using labels and variables defined in Section II-C-1. Once $(\beta, R3, \mathbf{B}, \mathrm{e})$ happens the workcell reaches State 300. This determination of succeeding states and their corresponding occurrence time is repeated until either a new decision node or the goal node is reached.

B. Example 2

Let us consider a workcell more complex than the one in Example 1: a three-machine and one-robot workcell with four types of parts, **A**, **B**, **C**, and **D**, each having one route, as adopted from [3]. The layout of the workcell is shown in Fig. 8. The sequence and processing times for each operation on each machine corresponding to each part type are given in Table IV. The automaton **SUPER** for this example consists of 1360 states and 3064 transitions. It was obtained in 115 s. Optimal

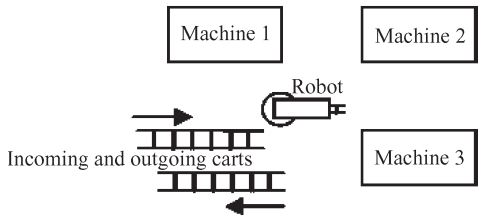

Fig. 8. Layout of the workcell.

TABLE IV
SEQUENCE OF OPERATIONS FOR PARTS **A**, **B**, **C**, AND **D**

Part Type	Operation Number	Device	Processing Time	Part Type	Operation Number	Device	Processing Time
	1	*R*1	5		f	*R*1	6
	2	*M*1	40		g	*M*1	212
	3	*R*1	3		h	*R*1	7
A	4	*M*2	100	**C**	i	*M*2	73
	5	*R*1	5		j	*R*1	4
	6	*M*3	36		k	*M*3	32
	7	*R*1	4		l	*R*1	5
	8	*R*1	5		m	*R*1	4
	9	*M*2	45		n	*M*3	55
	a	*R*1	3		o	*R*1	3
B	b	*M*1	65	**D**	p	*M*2	65
	c	*R*1	6		q	*R*1	5
	d	*M*3	98		r	*M*1	35
	e	*R*1	4		s	*R*1	5

TABLE V
SCHEDULING RESULTS FOR DIFFERENT BATCHES

Batch				Makespan	Computation Time (second)
A	**B**	**C**	**D**		
1	1	1	1	560	2.3
2	2	0	2	665	3.2
3	4	1	2	1244	12.1
5	5	5	5	2513	175

makespan for different batches are shown in Table V. One can note that since constructing **SUPER** is independent of the batch sizes, there is no need to change or reconstruct **SUPER** for each new batch. In [3], a problem of batch size of 1 (1, 1, 1, 1), one unit of each part types was formulated and solved on a Sun Sparc 20 computer in 67 s. The optimal makespan found for this batch was 560. As shown in Table V, the same result was obtained by our method. The time spent to find the optimal schedule for this batch was 2.3 s on our computer.

C. Example 3

The computation of the heuristic function for three different part-processing cases was explained in Section III-B. In the above two examples, however, only Case 1 was examined. The example in this Section, on the other hand, is introduced to illustrate the effectiveness of the proposed algorithm under Case 2, Case 3, as well as combinations of Case 1, Case 2, and Case 3. Let us assume that the workcell shown in Fig. 8 is capable to produce three part types **A**, **B**, and **C**. Part **A** has only one route for its production, i.e., Case 1. Part **B** has two routes but the required devices in both routes are the same, i.e., Case 2, and Part **C** has three routes and the required devices in each route

TABLE VI
SEQUENCE OF OPERATIONS FOR PARTS **A**, **B**, AND **C**

Part Type	Operation Number	Device	Processing Time	Part Type	Operation Number	Device	Processing Time
A				**B**			
One Route	1	*R*1	3	**Two Routes**	6	*R*1	3
	2	*M*1	20		7	*M*2	25
	3	*R*1	4		8	*R*1	4
	4	*M*2	40		9	*M*3	45
	5	*R*1	5		a	*R*1	5
C							
Three Routes	g	*R*1	3				
	h	*M*2	40				
	i	*R*1	5		b	*R*1	3
	j	*R*1	3		c	*M*3	25
	k	*M*3	30		d	*R*1	4
	l	*R*1	5		e	*M*2	40
	m	*R*1	3		f	*R*1	5
	n	*M*1	20				
	o	*R*1	5				

are different, i.e., Case 3. The sequence and processing times for each operation on each machine corresponding to each part type and each route are given in Table VI. The automaton **SUPER** for this example consists of 1773 states and 4520 transitions. It was obtained in 210 s.

In order to illustrate effectiveness of the heuristic function in reducing the number of nodes required to generate a solution, first, the search algorithm was executed assuming that the value of $\mathcal{H}$ is equal to zero for all nodes, i.e., not using the heuristic function. Next, solutions for the same batches were obtained using the proposed heuristic function. The results are given in Table VII.

Batches corresponding to Rows 1–4 in Table VII use only part type **B**, i.e., Case 2. As expected, the makespan found when $\mathcal{H}\#0$ is equal to the optimal one, when $\mathcal{H} = 0$. In Rows 5–8, the batches use part types **A** and **B**, i.e., Case 1 and Case 2, respectively. As expected, the search with the proposed heuristic function yielded the optimal solution. One notes that the percentage of node reduction significantly increases as the number of parts within the batch increases. As discussed in Section III-D, for Case 3, for most FMC configurations, the use of the heuristic function may yield the optimal makespan. However, there might be instances where the value derived by the heuristic function exceeds $\mathcal{H}^*$ and consequently, the search may not yield the optimal makespan, though it is likely to find acceptable solutions. To demonstrate this situation, batches corresponding to Rows 9–23 were simulated. The searches with the heuristic function yielded makespans either equal to or comparable to optimal values, while providing node reductions in the range of 38%–96%.

D. Discussion

In PN-based methods, deadlock detection is part of the scheduling process. Namely, as the reachability tree is generated, the deadlock states are designated to prune for further expansions as well as the paths leading to them. Thus, the time spent to obtain a deadlock-free schedule, for a given batch of parts, includes the time required to model the FMC as well as the generation of the reachability tree, deadlock detection, and

TABLE VII
SEARCH RESULTS FOR BATCHES

	Row	Batch			H = 0		H ≠ 0			Percentage of Node Reduction
		A	B	C	Number of Nodes	Make Span	Number of Nodes	Make Span	Computation Time (second)	
Cases 1 and 2	1		3		29	175	25	175	1.3	13.7
	2		5		57	273	29	273	2.3	49.1
	3		7		85	371	35	371	2.4	58.8
	4		9		113	469	39	469	2.7	65.4
	5	1	1		58	86	21	86	1.0	63.7
	6	1	2		254	140	113	140	5.0	55.5
	7	2	1		255	130	54	130	2.3	79.2
	8	2	2		901	167	240	167	9.6	73.3
Case 3 combined with Cases 1 and 2	9			3	272	48	28	54	1.0	89.7
	10			4	845	56	31	59	1.3	96.3
	11			5	2446	76	78	79	2.6	96.8
	12		1	1	93	77	46	77	2.1	50.5
	13		1	2	651	77	255	80	9.6	60.8
	14		2	1	404	126	249	126	10.7	38.4
	15		2	2	2501	126	495	126	19.4	80.2
	16	1		1	49	72	16	72	.6	67.3
	17	1		2	418	72	92	72	3.6	77.9
	18	2		1	231	121	73	121	3.7	68.3
	19	2		2	1517	121	170	127	6.5	88.7
	20	1	1	1	781	86	158	86	6.9	79.7
	21	2	1	1	2529	130	337	136	14.0	86.6
	22	1	2	1	2959	140	578	140	24.0	80.4
	23	1	1	2	3896	92	287	100	11.3	92.6

node evaluation. For further batches, this process must be repeated. In our proposed method, on the other hand, a deadlock-free search space (**SUPER**) is first constructed to represent the FMC behavior. Then, for a given batch, nodes are generated and evaluated. Since nodes are generated using **SUPER**, they are already deadlock free. Thus, the time to obtain a schedule consists of the time spent on creating **SUPER**, generating the nodes, and performing the evaluation. One can note that for the next batches of parts, no time will be spent creating the **SUPER**.

In Table II, it was shown that the number of nodes generated by our proposed approach is less than that of PN. The reduction is due to the fact that the constructed supervisor, employed by the search procedure, allows the generation of only deadlock-free nodes as opposed to the PN-based method, where the search procedure has to generate all possible nodes, designate deadlock nodes, and then proceed with nodes that are deadlock free. However, according to the above discussion, it would be difficult to conclude which approach is more efficient. In order to reach an indisputable conclusion, one would have to simulate a large number of FMC scenarios using all three approaches.

One must also note that in an automata-based control environment, the derivation of the supervisor is inherently necessary for PLC-based supervisory control, regardless of whether it is to be used for scheduling purposes as well or not. Furthermore, as the production proceeds, the time spent on constructing the **SUPER** is distributed among all future batches and thus, it becomes negligible.

V. CONCLUSION

This paper proposed a new automata-theory-based approach for deadlock-free scheduling of FMCs. The concept of supervisory-control theory is extended to the scheduling problem by augmenting the automata with time labels. An A^* search algorithm that employs a heuristic function is employed to search for the optimal deadlock-free schedules subject to makespan as the performance criterion.

Three different part-processing requirements were considered. In Case 1, where each part has a unique route to be produced, the search algorithm yields the optimal solution. For Case 2, where parts have multiple routes and required devices on each route are the same, the search also yields the optimal solution. In Case 3, where each route requires different devices, the search procedure may not yield the optimal solution, but simulations have shown promising results.

The major advantage of using the constructed supervisor is that it is deadlock free by construction and is independent of the batch information. This allows the search procedure to generate only deadlock-free nodes and prevent repeating the deadlock-detection phase for further batches of parts, which is not the case in PN-based or mathematical-programming-based deadlock-free scheduling approaches.

REFERENCES

[1] K. B. Baker, *Introduction to Sequencing and Scheduling*. New York: Wiley, 1974.
[2] M. Pinedo, *Scheduling Theory: Algorithms and Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1995.

[3] S. E. Ramaswamy and S. B. Joshi, "Deadlock-free schedules for automated manufacturing workstations," *IEEE Trans. Robot. Autom.*, vol. 12, no. 3, pp. 391–400, Jun. 1996.
[4] P. B. Luh and D. J. Hoitomt, "Scheduling of manufacturing systems using the Lagrangian relaxation technique," *IEEE Trans. Automat. Contr.*, vol. 38, no. 7, pp. 1066–1079, Jul. 1993.
[5] F.-S. Hsieh and S.-C. Chang, "Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems," *IEEE Trans. Robot. Autom.*, vol. 10, no. 2, pp. 196–209, Apr. 1994.
[6] H. S. Yan, N. S. Wang, X. Y. Cui, and J. G. Zhang, "Modeling, scheduling and control of flexible manufacturing systems by extended high-level evaluation Petri nets," *IIE Trans.*, vol. 29, no. 2, pp. 147–158, 1997.
[7] Y. Arzi and L. Iaroslavitz, "Neural network-based adaptive production control system for a flexible manufacturing cell under a random environment," *IIE Trans.*, vol. 31, no. 3, pp. 217–230, 1999.
[8] W. Guohua and P.-C. Yen, "A fuzzy logic system for dynamic job shop scheduling," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, Tokyo, Japan, 1999, pp. 546–551.
[9] H. H. Xiong, M. Zhou, and C. N. Manikopoulos, "Scheduling flexible manufacturing systems based on timed Petri nets and fuzzy dispatching rules," in *Proc. IEEE Symp. Emerging Technologies and Factory Automation*, Paris, France, 1995, pp. 309–315.
[10] S. R. Jernigan, S. Ramaswamy, and K. S. Barber, "On-line scheduling using a distributed simulation technique for intelligent manufacturing systems," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, Vancouver, Canada, 1995, pp. 2159–2164.
[11] Y. Arzi and L. Iaroslavitz, "Operating an FMC by a decision-tree-based adaptive production control system," *Int. J. Prod. Res.*, vol. 38, no. 3, pp. 675–697, 2000.
[12] L. C. Rabelo, A. Jones, and Y. Yih, "Development of a real-time learning scheduler using Reinforcement Learning concepts," in *Proc. IEEE Int. Symp. Intelligent Control*, Columbus, OH, 1994, pp. 291–296.
[13] T. O. Boucher, A. Yalcin, and T. Tai, "Dynamic routing and the performance of automated manufacturing cells," *IIE Trans.*, vol. 32, no. 10, pp. 975–988, 2000.
[14] A. J. Malo-Tamayo, D. Gavino-Contreras, and A. Ramirez-Trevino, "Petri net based control for the dynamic scheduling of a flexible manufacturing cell," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, San Diego, CA, 1998, pp. 553–557.
[15] S. C. Park, N. Raman, and M. J. Shaw, "Adaptive scheduling in dynamic flexible manufacturing systems: A dynamic rule selection approach," *IEEE Trans. Robot. Autom.*, vol. 13, no. 4, pp. 486–502, Aug. 1997.
[16] I. Sabuncuoglu and S. Karabuk, "A beam search-based algorithm and evaluation of scheduling approaches for flexible manufacturing systems," *IIE Trans.*, vol. 30, no. 2, pp. 179–192, 1998.
[17] I. B. Abdallah, H. EI Maraghy, and T. El Mekkawy, "An efficient search algorithm for deadlock-free scheduling in FMS using Petri nets," in *Proc. IEEE Int. Conf. Robotics and Automation*, Leuven, Belgium, 1998, pp. 1793–1798.
[18] 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.
[19] A. R. Moro, H. Yu, and G. Kelleher, "Hybrid heuristic search for the scheduling of flexible manufacturing systems using Petri nets," *IEEE Trans. Robot. Autom.*, vol. 18, no. 2, pp. 240–245, Apr. 2002.
[20] H. H. Xiong, M. Zhou, and R. J. Caudil, "A hybrid heuristic search algorithm for scheduling manufacturing systems," in *Proc. IEEE Int. Conf. Robotics and Automation*, Minneapolis, MN, 1996, pp. 2793–2797.
[21] X. Cao and Y. Ho, "Models of discrete event dynamic systems," *IEEE Control Syst. Mag.*, vol. 10, no. 4, pp. 69–76, Jun. 1990.
[22] P. J. G. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event systems," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.
[23] M. Zhou and K. Venkatesh, *Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach*. Singapore: World Scientific, 1998.
[24] S. Cavalieri and O. Mirabella, "A PN-based scheduler for a flexible semiconductor manufacturing system," in *Proc. IEEE Conf. Emerging Technologies and Factory Automation*, Kauai, HI, 1996, vol. 2, pp. 724–729.
[25] C. W. Cheng, T. H. Sun, and L. C. Fu, "Petri-Net based modeling and scheduling of a flexible manufacturing system," in *Proc. IEEE Int. Conf. Robotics and Automation*, San Diego, CA, 1994, vol. 1, pp. 513–518.
[26] U. Syed, A. Robbi, and M. Zhou, "Anticipatory real-time scheduling in manufacturing cell design," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, Vancouver, Canada, 1995, pp. 4131–4136.
[27] B. C. Damasceno and X. Xie, "Scheduling and deadlock avoidance of a flexible manufacturing system," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, San Diego, CA, 1998, vol. 1, pp. 564–569.
[28] M. Der Jeng and S. Chang Chen, "Heuristic search based on Petri net structures for FMS scheduling," *IEEE Trans. Ind. Appl.*, vol. 35, no. 1, pp. 196–202, Jan.–Feb. 1999.
[29] H. H. Xiong and M. Zhou, "Deadlock-free scheduling of an automated manufacturing system based on Petri nets," in *Proc. IEEE Int. Conf. Robotics and Automation*, Albuquerque, NM, 1997, pp. 945–950.
[30] S. C. Lauzon, A. K. L. Ma, J. K. Mills, and B. Benhabib, "An implementation methodology for the supervisory controller for flexible manufacturing workcells," *SME, J. Manuf. Syst.*, vol. 16, no. 2, pp. 91–101, 1997.
[31] A. Ramirez-Serano, 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.
[32] A. Ramirez Serano, S. C. Zhu, and B. Benhabib, "Moore automata for flexible routing and flow control in manufacturing workcells," *J. Auton. Robots*, vol. 9, no. 1, pp. 59–69, 2000.
[33] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. New York: Addison-Wesley, 1979.
[34] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Los Angeles, CA: Addison-Wesley, 1984.

Hamid Reza Golmakani received the B.Sc. and M.Sc. degrees in industrial engineering from Amirkabir University and Sharif University, Iran, in 1988 and 1991, respectively. He is working toward the Ph.D. degree in mechanical and industrial engineering at the University of Toronto, Toronto, ON, Canada.

From 1991 to 2000, he has been a Faculty Member in the department of Industrial Engineering, Amirkabir University. His research interests include production planning, scheduling, and control of flexible manufacturing systems.

James K. Mills (S'81–M'82) received the B.Sc. degree from the University of Manitoba, Winnipeg, MB, Canada, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada.

He is a Professor in the Department of Mechanical and Industrial Engineering, University of Toronto. His research interests include control of multirobots, smart structure design and control, mechatronic design, localization, development of fixtureless assembly technology, design and control of high-speed machines, convex control design, three-dimensional microelectromechanical system (3-D MEMS) assembly, etc. He has published over 250 journals and conference papers and supervised over 50 Masters, Ph.D., and Post-Doctoral students. He has been an Invited Visiting Professor at the Centre for Artificial Intelligence and Robotics in Bangalore, India, the Hong Kong University of Science and Technology, City University of Hong Kong, as well as the Chinese University of Hong Kong.

Beno Benhabib received the B.Sc. degree from Bogazici University, Istanbul, Turkey, the M.Sc. degree from Technion, Israel, and Ph.D. degree from University of Toronto, Toronto, ON, Canada.

Since 1986, he has been a Professor in the Department of Mechanical and Industrial Engineering as well as Electrical and Computer Engineering at the University of Toronto. His research interests are in the field of autonomous systems.