Process Dynamics-Aware Flexible Manufacturing for Industry 4.0

Michael Balszun[1], Clara Hobbs[2], Enrico Fraccaroli[2], Debayan Roy[1], Franco Fummi[3], Samarjit Chakraborty[2]
[1]TU Munich, Germany, [2]UNC Chapel Hill, USA, [3]University of Verona, Italy

***Abstract*— This paper studies the following basic flexible manufacturing problem: Given N machines that can perform the same job on a production item (e.g., drilling or tapping) but with different capabilities (e.g., energy requirements and speeds), what is an optimal *schedule* for the job on these machines? While this is a well-studied problem, the main innovation this paper introduces is the explicit modeling of the underlying process dynamics—i.e., the physical interaction of the item and the machine—using differential equations. The resulting scheduling problem is in a *hybrid systems setting* that involves determining the transition times between states, where the system evolution in each state is defined by differential equations. To the best of our knowledge, such a cyber-physical systems (CPS) oriented approach to machine scheduling has not been studied before, although it lies at the core of flexible manufacturing in Industry 4.0. We believe that this new formulation might lead to a renewed interest in machine scheduling problems, but now in a hybrid/CPS-oriented setting.**

I. Introduction

Flexible manufacturing systems in Industry 4.0 require intelligently adapting multiple machines to meet changing requirements. How do these adaptation decisions depend on the underlying dynamics of the machines involved? To answer this, we study the scheduling problem shown in Fig. 1. The question is: given a raw material that needs to be machined into a final product, and a set of *machines* or multiple *modes of operation* of a single machine, how much time should the item spend in each machine or mode? Each machine has different characteristics and capabilities (e.g., hardness of its drill bit, maximum input power, and consequently processing times and energy requirements). In addition to obtaining the finished product, the goal is to optimize associated metrics like completion time and energy consumption. Focusing on multiple performance metrics and *automatically* adapting the manufacturing process is the hallmark of Industry 4.0.

The core problem here is that of *machine scheduling*, which has been widely studied for many years ([1], [2], [3], just to name a few), both in the manufacturing and job scheduling communities. However, almost all previous studies have been restricted to an inaccurately reduced setting where the remaining execution time of the job decreases linearly over time as it is processed. In contrast to this traditional approach, in this paper, we study a machine scheduling problem following a *cyber-physical systems* approach. The goal is to explicitly *model the physical dynamics* of how a machine interacts with the material. Thus, instead of considering only processing times that decrease linearly, we model the transformation of the material into a finished product using a set of differential equations, to more accurately capture the physical dynamics of the manufacturing process. In such a hybrid model, discrete state transitions represent how the material shifts from one machine to another (or how the configuration of a machine changes), and within a state, the continuous transformation of the material by the machine is modeled using differential equations. Using this model, we can formulate a scheduling problem to determine the state transition *times* that optimize metrics like completion time and energy cost.

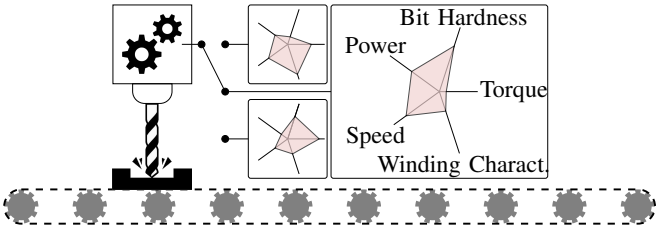

Fig. 1. Process dynamics change with different modes.

Proposed solution: We assume that we can switch between machines only at uniformly-spaced discrete instants of time. A node at depth k in the resulting search tree represents a partial solution comprising machine transitions until the k-th time step. To keep the search tractable, we perform predictive pruning. We compare nodes (or partial solutions) based on the state of the job, the time elapsed, and the available machine transitions, and consider only the sub-trees originating from the promising nodes for further exploration. Through this procedure, we generate a set of one or more Pareto-optimal solutions. In our experiments, this technique can synthesize results orders of magnitude faster than an exhaustive search (in a few milliseconds on average) and in particular avoid the long worst case runtimes typical for exhaustive searches. At the same time, the solutions are only up to 4% worse (less than 1% in 500 out of 564 cases) in terms of the production objectives, i.e., job completion time and energy consumption.

Paper organization: We first present a brief summary of related work. Then, we formally introduce the scheduling problem in Sec. II. We formulate a thread tapping process in Sec. III to illustrate the problem. Next, we explain our optimization approach in Sec. IV, and present results of an experimental evaluation using the tapping process in Sec. V. Finally, Sec. VI provides concluding remarks.

A. Related work

Scheduling is a well-studied topic in adaptive manufacturing. Different scheduling techniques for Hybrid Flow Shop (HFS) problems, where production happens in a fixed num-

ber of predefined stages, each consisting of multiple identical machines, have been discussed, e.g., in [3]. In scenarios where machines are used in different orders, the scheduling strategy must ensure safety properties such as deadlock freedom [4]. Another commonly investigated issue is adapting the schedule in the presence of machine breakdown and accommodating scheduled maintenance downtime [5]. In order to quickly evaluate the impact of various online changes to a given schedule (e.g., due to the arrival of new high priority orders), [6] presents a real-time simulation model for flexible manufacturing systems. Similar scheduling problems have also been studied in industrial process control [7], and in the design of hardware/software control systems, e.g., in the automotive domain [8], [9], [10].

Unfortunately, all of these techniques model the individual processing steps as fixed-length work units, and at most consider different setup costs when a machine has to switch tasks types. They do not model the steps themselves as continuous-time *dynamical processes*, whose behavior could change at arbitrary points in time. A hybrid model for flexible manufacturing systems has been described in [11]. However, the differential equations describe the flow of parts through the factory, and not the individual manufacturing steps themselves, as we do in this paper.

There is also a line of research that applies social- and stochastic-based optimization techniques to manufacturing job scheduling. For instance, the work in [12] tackles the environmental aspects (power consumption, and tardiness) in the job shop scheduling problem by applying a multi-objective grey wolf optimizer. Similarly, [13] presents a multi-objective particle swarm optimization approach for optimizing the on-off behaviour of one computerized numerical control machine. Both approaches generate a set of Pareto-optimal solutions, offering a trade-off between energy consumption and process time. But again, they treat the individual jobs as fixed-length items.

Beyond flexible manufacturing, process control and industrial control (including the automotive domain), the real-time systems community also has a long history of studying task scheduling, including in cyber-physical systems [14]. While such literature rarely addresses flexible manufacturing scenarios directly (even though e.g., [15] has investigated the scheduling of CPU tasks for a flexible manufacturing setup), the employed techniques can also be applied to the scheduling of manufacturing jobs. However, they do not permit the change in remaining execution times of jobs (or other related metrics) to be defined by differential equations, as is the case in our paper. Some related scheduling problems, albeit not considering the rich dynamics that we study here, include [16] that addresses the dynamic scheduling of tasks under real-time constraints, and works like [17] and [18] that target the problem of multi-mode scheduling with similarities to switching between different production modes in manufacturing systems. Our setup also has similarities with hybrid/multi-mode systems studied in [19] where the goal was safety verification, whereas we consider optimization and a very different problem context.

II. Dynamics-Aware Scheduling Problem

Let us consider a manufacturing process where a machine performs some operations on a production item. In this paper, we study processes for which the dynamics can be captured using a set of linear differential equations of the form

$$\dot{x}(t) = \boldsymbol{A}\, x(t) + \boldsymbol{B}\, u. \tag{1}$$

Here, the time-varying vectors $x(t)$ and u are the states of and the inputs to the process (comprising both the item being machined and the machine itself) respectively, while the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are process constants. Note that this model may not only describe the physical system itself, but can also include any form of linear controller needed to ensure stability of the system, and to guide the process to the target state. Consequently, u may not only be a physical system input, but might contain fixed control inputs such as a reference point.

In flexible manufacturing, we may have several *machines* that can perform the same operations but with different capabilities, e.g., speed and energy usage. A process is assumed to have n machines available denoted $\mathcal{M}_1, \ldots, \mathcal{M}_n$, each with different process dynamics of the form in Eq. (1), with constants denoted by $\boldsymbol{A}_i$, $\boldsymbol{B}_i$ and u_i.

The manufacturing processes in this setting are controlled by a computer, allowing us to choose the best machine at any time to optimize production efficiency. Thus, we can model the manufacturing process as a switched dynamical system

$$\dot{x}(t) = \boldsymbol{A}_{\sigma(t)}\, x(t) + \boldsymbol{B}_{\sigma(t)}\, u_{\sigma(t)}, \tag{2}$$

where $\sigma(t)$ is the machine selection function, with $\sigma(t) = i$ indicating that machine $\mathcal{M}_i$ is selected at time t.

The above formulation does not describe the effect that the machine transitions themselves can have on the state. This is sufficient if different machines only differ in the control parameters or the effects are negligible. However, if the switch involves e.g. the change of a tool or handing over the object to a completely different machine, this effect must be modeled too. In this work, we model a switch at time ts as an atomic operation, represented by a function $x_{ts+} = \mathcal{F}_{ji}(x_{ts-})$ that depends on the state just before the switch x_{ts-} and the selected modes before and after the switch $j = \sigma(ts-)$ and $i = \sigma(ts+)$ respectively.

In essence, a flexible manufacturing process thus defined is a hybrid system. The process state evolves in continuous time, while the process control software can cause discrete transitions between dynamical laws by changing the operating machine.

In addition to reaching a target region X_{tgt} from an initial state x_{ini}, a typical production objective is to minimize the processing cost e, which can e.g. consist of processing time and energy. Similarly to the state evolution, we model the cost evolution while a particular mode is active as a differential equation $\dot{e} = P_i\big(x(t), u_i\big)$, and the cost for switching from mode j to mode i at time ts as an opaque function $\mathcal{E}_{ji}(x_{ts-})$. In many manufacturing processes, the available machines present a trade-off between time and

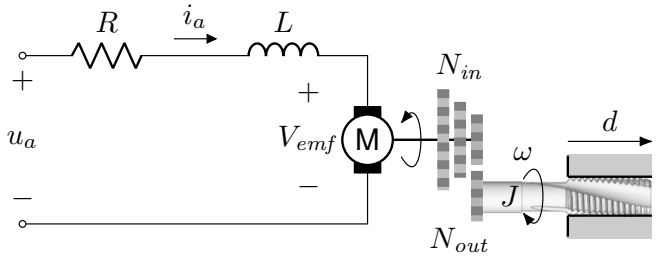

Fig. 2. Schematic of the tapping process.

energy. In such cases, the scheduling problem can be treated as a multi-objective optimization problem, yielding several Pareto-optimal solutions. **The goal in this paper is to obtain these solutions efficiently.**

III. Illustrative Problem: Tapping

As a motivational example, we study the thread tapping process shown in Fig. 2, where a milling machine equipped with a tap cuts a thread inside a pre-drilled hole. Typically, such a machine has different modes of operation that a computer can select. For instance, the machine could run at different angular speeds and torques by changing the gear setting, yielding different process dynamics.

The tapping process can be modeled based on the differential equations governing a direct current (DC) motor assuming a homogeneous magnetic field [20]. We extend these equations to account for Coulomb friction depending on the depth of the tap. The input of the system is the voltage applied to the armature winding.

On the electrical side, the resistance R models how the armature limits the maximum current of the motor alongside the back-EMF voltage V_{emf}, while the inductance L models how it limits the current over time. The motor's shaft is connected to n gears, each having $N_{in}^{(1)}, N_{in}^{(2)}, \ldots, N_{in}^{(n)}$ teeth. The rotation is transmitted to an output gear with N_{out} teeth. Given the input gear $N_{in}^{(i)}$, the gear ratio is computed as $G_r^{(i)} = N_{in}^{(i)}/N_{out}$. For a given $G_r^{(i)}$, the DC motor dynamics are governed by the following differential equations:

$$K_t\, i_a(t) = J\,\frac{d\omega}{dt} + K_d\,\omega(t) + F_d\, G_r^{(i)}\, d(t) + G_r^{(i)}\, F_s \quad (3)$$

$$v_a(t) = R\, i_a(t) + L\,\frac{di_a}{dt} + K_e\,\omega(t). \quad (4)$$

Here, K_e, K_t, K_d, F_d, and F_s denote the back-EMF, motor torque, damping, dynamic hole friction, and static hole friction constants, respectively. The constant J gives the rotational inertia. At time t, the variables $v_a(t)$, $\omega(t)$, $i_a(t)$, and $d(t)$ represent the input voltage, the angular speed, the armature's current, and the depth of the tap. The term $F_d G_r^{(i)} d(t)$ in Eq. (3) accounts for the frictional force that depends on the tap's depth. The depth d is computed based on the linear relation between a single complete spindle rotation and the thread slope T_s, measured in millimeters per revolution. For $G_r^{(i)}$, the relation can be captured using the differential equation:

$$\frac{dd}{dt} = \frac{T_s G_r^{(i)}}{2\pi}\omega(t). \quad (5)$$

By letting the state vector $x = [\omega\ i_a\ d]^T$ and the input vector $u = [v_a\ F_s]^T$, we can combine Eq. (3), (4), and (5) to derive the following matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ for Eq. (1):

$$\boldsymbol{A}_i = \begin{bmatrix} -\frac{K_d}{J} & \frac{K_t}{J} & -\frac{F_d\, G_r^{(i)}}{J} \\ -\frac{K_e}{L} & -\frac{R}{L} & 0 \\ \frac{T_s\, G_r^{(i)}}{2\pi} & 0 & 0 \end{bmatrix} \quad \boldsymbol{B}_i = \begin{bmatrix} 0 & -\frac{G_r^{(i)}}{J} \\ \frac{1}{L} & 0 \\ 0 & 0 \end{bmatrix} \quad (6)$$

For each gear ratio $G_r^{(i)}$, we obtain different process dynamics, distinguished by $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$. For n different input gears, the process can switch between n different dynamics. At time t, the process constants are given by $\boldsymbol{A}_{\sigma(t)}$ and $\boldsymbol{B}_{\sigma(t)}$ where $\sigma(t) \in \{1, 2, \ldots, n\}$. As explained in Sec. II, our main research question is: **What is an optimal way to switch between these different dynamics?**

A. Numerical example

Let us now consider a numerical example exemplifying how each machine setting might offer a different trade-off between production time and cost. Let us assume that the motor initially has zero rotational speed, zero current through the armature, and zero tapping depth. The target depth is $20\,\mathrm{mm}$. The values for the process constants are given by:

$$\begin{gathered} v_a = 9.6 \quad L = 2.5 \times 10^{-4} \quad R = 1 \quad J = 0.1 \\ T_s = 1 \quad K_d = 0.25 \quad K_e = 1 \quad K_t = 1 \\ F_d = 0.01 \quad F_s = 0.05 \end{gathered} \quad (7)$$

Given these constants and the state-space model in Eq. (6), it is possible to simulate the tapping process.

Let us consider two settings for the machine corresponding to the gear ratios $G_r^{(1)} = 15$ and $G_r^{(2)} = 30$. As shown in Fig. 3, using $G_r^{(1)}$ and $G_r^{(2)}$, the time in which the target depth can be reached is approximately $1.5\,\mathrm{s}$ and $1.1\,\mathrm{s}$, respectively. Thus, the process is approximately $36\,\%$ slower with the former setting than the latter, but it draws less power. If we measure the production cost in terms of the energy spent on the process, $G_r^{(1)}$ is more cost-saving compared to $G_r^{(2)}$. This shows that there is a trade-off opportunity between the cost and the time that needs to be explored.

Consider, for example, that this tapping process has a production deadline of $1.3\,\mathrm{s}$. This deadline can be met by just using $G_r^{(2)}$. However, we can ask two questions here: (i) is that the only choice? (ii) is that optimal in terms of the energy spent? The computer could use a combination of both modes. As the production objectives (e.g., the cost and the speed) can conflict with each other, there might be several Pareto-optimal solutions. Depending on the production requirements, one such solution needs to be used for the process.

In a tapping process, the dynamic friction increases with depth. As the friction counterbalances the motor torque, the spindle speed approaches zero. In Fig. 3, we can see that the

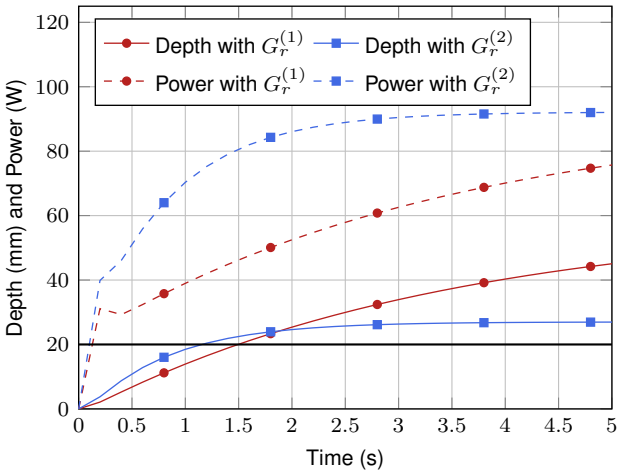

Fig. 3. Simulating the tapping process with two gear settings, i.e., $G_r^{(1)}$ and $G_r^{(2)}$. Solid lines identify the depth in millimetres achieved by each gear setting, while dashed lines identify the power drawn in watts (W).

maximum tapping depth with $G_r^{(2)}$ is around $27\,\mathrm{mm}$. With $G_r^{(1)}$, we can tap deeper because a higher rotational torque can be extracted with a lower gear ratio. If the target tapping depth is increased to $40\,\mathrm{mm}$, a straightforward solution is to use $G_r^{(1)}$ for the entire tapping process. However, we can also use $G_r^{(2)}$ at lower depths and then switch to $G_r^{(1)}$ to reduce the process time at the cost of more energy. This again demonstrates that there is a scope for schedule optimization in a manufacturing process based on the production requirements.

Observe in Fig. 3 that at lower depths, the difference in tapping speed between $G_r^{(1)}$ and $G_r^{(2)}$ is larger (given by the slope of the depth curves), while the difference in power drawn is smaller. Thus, it becomes increasingly inefficient to use $G_r^{(2)}$ as the tap's depth increases. Using this relation, we can constrain the schedule optimization problem so that at any time during the tapping process, we can only switch to a lower gear. Such a constraint can reduce the complexity of the optimization problem significantly.

IV. Proposed Optimization Strategy

In order to determine a Pareto-optimal set of switching sequences, we propose to discretize the process model in Sec. II using a fixed time step h. Machine switches are only allowed (but not required) between steps, and the same machine is used for the duration of a step.

A. Discretization

The input u_i is a fixed property of the machine $\mathcal{M}_i$, and is thus constant during each time step. Therefore, the discrete state change when applying machine $\mathcal{M}_i$ can be modeled as

$$\begin{aligned} x[k+1] &= F_i(x[k]), \quad \text{where} \\ F_i(x) &= \Phi_i x + \Gamma_i u_i, \quad \Phi_i = e^{\boldsymbol{A}_i h}, \quad \text{and} \\ \Gamma_i &= \int_0^h e^{\boldsymbol{A}_i s} \boldsymbol{B}_i \, ds. \end{aligned} \tag{8}$$

Further, we assume that we can write the cost function as

$$e[k+1] = e[k] + E_i(x[k]), \tag{9}$$

where $E_i(x)$ is always non-negative in all dimensions, and thus $e[k]$ monotonically increases in all dimensions over time. Time can either be modeled explicitly as a dimension of e, or calculated (more efficiently) as $k \cdot h$ (plus the necessary time for any switches). For the example presented in Sec. III, the cost is time and energy, and we compute energy as the integral over voltage times the current, which can be expressed as

$$E_i(x) = \int_0^h u_i^T \mathcal{N} \mathcal{X}_i(t, x)\, dt. \tag{10}$$

where $\mathcal{X}(t, x)$ is the state evolving according to Eq. (1) from the state x at the beginning of a time step, and $\mathcal{N} = \left[\begin{smallmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \end{smallmatrix}\right]$. Solving the integral leads to Eq. (11) (we omit the intermediate steps for space reasons):

$$\begin{aligned} E_i(x) &= u_i^T \mathcal{S}_i x + u_i^T \mathcal{R}_i u_i \\ \text{with } \mathcal{S}_i &= \mathcal{N} A_i^{-1} \left(e^{A_i h} - I\right), \\ \mathcal{R}_i &= \mathcal{N} A_i^{-1} \left(A_i^{-1} e^{A_i h} - A_i^{-1} - I\, h\right) B_i \end{aligned} \tag{11}$$

If the mode switch has non-negligible effects on cost and process state (see Sec. II), we can add these effects in $F_i(x)$ and $E_j(x)$, which would then depend not only on the current mode $\mathcal{M}_i$, but also the previous mode $\mathcal{M}_j$. So $F_{ji}(x) = F_i(\mathcal{F}_{ji}(x))$ with $\mathcal{F}_{jj}(x)$ being the identity function and $E_{ji}(x) = E_i(\mathcal{F}_{ji}(x)) + \mathcal{E}_{ji}(x)$ with $\mathcal{E}_{jj} = \mathbf{0}$. Additionally, we need to model time explicitly as part of the cost e in this case, as each transition may take different amounts of time. However, in order to keep the notation short, we will use the notation $F_i(x)$ and $E_i(x)$ in the rest of the paper also for the generalized cases.

B. Pareto optimization

Using the above formulation, we now construct a search tree over the possible machine selection schedules. Each tree node $\widetilde{\omega}$ represents a potential solution, i.e. a sequence of machine selections $\mathcal{I}$ that may or may not drive x into X_{tgt}. $\mathcal{I}$ corresponds to the sequence of selected branches in the tree leading to the current node. We represent every node $\widetilde{\omega}_a$ as a tuple containing:

1) the sequence of selected branches (selected machines) $\mathcal{I}_a = [i_1, i_2, \ldots, i_k]$ leading to the current node;
2) the state $x_a = F_{i_k}\left(F_{i_{k-1}}(\ldots(F_{i_1}(x_{ini}))\right)$ we get after applying the sequence of state transformations;
3) the accumulated cost of those transitions $e_a = \sum_{l=1}^{k} E_{i_l}(x[l])$.

A node $\widetilde{\omega}_a$ in this tree could be uniquely described by $\mathcal{I}_a$, and both state and cost could be recomputed from that sequence and the initial state. However, creating and evaluating a new node can be done much more efficiently if we cache x and e, needing only to compute the difference from the parent. This is especially beneficial because the algorithm usually needs to create multiple child nodes for each $\widetilde{\omega}$, which can share the first $k-1$ computation steps for e and x with the parent. So, given a node $\widetilde{\omega}_a = \langle I_a, x_a, e_a \rangle$,

Algorithm 1: Search-based Pareto optimization

Input : The set of machines $\{\mathcal{M}_1, \dots, \mathcal{M}_n\}$
Output : The set of Pareto-optimal solutions Ω

1 Initialize the empty set of Pareto-optimal solutions (Ω);
2 Initialize list of partial solutions with initial machine ($\widetilde{\Omega}$);
3 **while** *There are partial solutions in* $\widetilde{\Omega}$ **do**
4 Create a new, empty list of partial solution ($\widetilde{\Omega}_{new}$);
5 **forall** *partial solutions* $\widetilde{\omega}$ *in* $\widetilde{\Omega}$ **do**
6 **for** *machine* $\mathcal{M}_i$ *in* $\{\mathcal{M}_1, \dots, \mathcal{M}_n\}$ **do**
7 Compute new partial solution $\widetilde{\omega}_{new}$ by applying machine $\mathcal{M}_i$ for one time step;
8 Add the new partial solution $\widetilde{\omega}_{new}$ to the list $\widetilde{\Omega}_{new}$;
9 Remove partial solutions $\widetilde{\omega}$ from $\widetilde{\Omega}_{new}$, that are dominated by those already in the set Ω;
10 **if** *using the heuristic* **then**
11 Prune partial solutions from $\widetilde{\Omega}_{new}$ based on heuristic criteria;
12 Move complete solutions from $\widetilde{\Omega}_{new}$ to Ω;
13 Replace $\widetilde{\Omega}$ with remaining partial solutions from $\widetilde{\Omega}_{new}$

applying machine $\mathcal{M}_i$ for one time step generates a child node $\widetilde{\omega}_b = \langle [\mathcal{I}_a, i], F_i(x_a), E_i(x_a) + e_a \rangle$.

A node $\widetilde{\omega}$ in the tree represents a *full solution* ω if and only if $x \in X_{tgt}$, and we call all other nodes *partial solutions*. Our goal is to find the set of full solutions Ω whose cost is not dominated by any other solution, meaning that for any $\omega_p \in \Omega$, there does not exist any other ω_q whose cost is smaller in all dimensions (e.g., time and energy). As the cost monotonically increases with each step, a node ω_{n+1} that is a child of ω_n will always be dominated by ω_n. Thus, any node ω_n that represents a full solution is a leaf node in the search tree. Furthermore, any node $\widetilde{\omega}_n$ corresponding to a partial solution whose cost is dominated by a known full solution can also never have a descendant that will be part of the Pareto front, and thus warrants no further exploration.

Alg. 1 shows the procedure for finding the Pareto front based on the principles discussed so far. Note that in this basic algorithm, we assume that we may apply any machine in any reachable state $x \notin X_{tgt}$. In practice, possible transitions between machines might be further restricted, leading to more nodes becoming leaves without being a complete solution. Alg. 1 is guaranteed to terminate if and only if 1) any possible sequence of machines will eventually drive the state towards X_{tgt}, or if 2a) at least one full solution is found and 2b) the cost for any sequence of machines is strictly monotonically increasing without converging to a level that is not dominated by that solution. In particular, condition 2a) can often be achieved by bootstrapping the algorithm, for instance, with a naïve solution that uses only one machine. Furthermore, condition 2b) is also reasonable in the majority of real-world problems. Practically, however, theoretical termination of the algorithm is less of a concern than termination within some reasonable time limit, as—in the worst case—the size of $\widetilde{\Omega}$ increases exponentially with each iteration.

This concern of practical termination is tightly coupled to the question of choosing an appropriate discretization time h, which is a trade-off between runtime and accuracy. A shorter h will cause deeper decision trees and, thus, exponentially more nodes to evaluate (when using a naïve breadth-first search), while a longer h results in coarser sampling of the solution space and thus less optimal solutions. One can think of some situations where a good h value is known *a priori*, e.g., based on previous trial-and-error runs. However, due to the exponential growth (at least for an exhaustive search), it is still hard to estimate the actual runtime *a priori*.

Instead, we propose a more robust approach, where the algorithm is first run with a long discretization time h^0, and then re-run iteratively with a shorter h^r. The process is repeated until either no significant improvements are seen, the runtime reaches a predefined maximum, or h^r reaches a predefined minimum. By choosing $h^{r+1} = h^r/2$, the runtime overhead of the previous iterations compared to that of the current iteration is minimized. For instance, in our test cases (see Sec. V), their combined runtime only comprised about 10–15% of the total runtime.

C. Problem-specific extensions and heuristics

If the exact problem structure and its constraints are known, as in our motivational example from Sec. III, the general algorithm can be further fine-tuned to improve its runtime. First, even if the optimal switching times are unknown, the order of selected machines may follow a known pattern. For example, in the tapping process, friction increases with depth, so an optimal solution will always switch from higher to lower gears. In other cases we might want to switch from a more aggressive control algorithm to a slower one when close to the target. For our example, this means the loop in line 6 need not iterate the full set of machines from $\mathcal{M}_1$ to $\mathcal{M}_n$, but can start at the currently used machine $\widetilde{\omega}.I[end]$ (assuming the machines are sorted in decreasing order of gear ratios). This constraint turns the exponential growth of nodes in each step into a polynomial growth, i.e., $\mathcal{O}(|\widetilde{\Omega}_{new}|) = \mathcal{O}(k^{n-1})$ instead of $\mathcal{O}(n^k)$, where n is the number of machines and k is the number of steps. Intuitively, this is because with a fixed order of machines, at most $n-1$ switches can happen and thus there are only $\mathcal{O}\left(\binom{k-1}{n-1}\right)$ possible switching sequences for k steps, which is $\mathcal{O}(k^{n-1})$.

As a second extension, we can apply heuristics to prune branches by comparing nodes with other partial solutions instead of waiting for full solutions to be found. In general, a partial solution $\widetilde{\omega}_a$ is dominated by $\widetilde{\omega}_b$ if and only if each full solution that is a descendant of $\widetilde{\omega}_a$ is dominated by at least one descendant of $\widetilde{\omega}_b$. However, under practical runtime constraints, it can be beneficial to use a non-optimal heuristic that can be run with a smaller discretization time (see our experimental results in Sec. V). To that end, we prune partial solutions not only when they are dominated by full solutions, but also when they are dominated by other partial solutions. Our heuristic determines the dominance between partial solutions by comparing not only the actual costs (e.g., time and energy), but also the distance to X_{tgt} and the remaining *degrees of freedom*. That is, in our case

study, the system can only switch from machine $\mathcal{M}_i$ to $\mathcal{M}_j$ if $i \leq j$, so if the current machine of the partial solution $\widetilde{\omega}_a$ is lower than that of $\widetilde{\omega}_b$, we say that $\widetilde{\omega}_a$ has more degrees of freedom. Eq. 12 summarizes this heuristic used in *prune* (Alg. 1):

$$\begin{aligned} \widetilde{\omega}_a \preceq \widetilde{\omega}_b \iff & \; dist(x_{tgt}, \widetilde{\omega}_a.x) \leq dist(x_{tgt}, \widetilde{\omega}_b.x) \\ \wedge & \; \widetilde{\omega}_a.I[end] \leq \widetilde{\omega}_b.I[end] \\ \wedge & \; \widetilde{\omega}_a.e \preceq \widetilde{\omega}_b.e \end{aligned} \tag{12}$$

V. Experimental Setup and Results

In this section, we validate our proposed scheduling strategy by applying it to the tapping process from Sec. III. We wrote a C++ implementation of the problem-specific version of Alg. 1 with and without the pruning heuristic (called *Heuristic* and *Exhaustive* in this section respectively), and compared their output and runtime for multiple variations of the tapping process. We discuss details of our experimental setup next, then present an analysis of the results.

Although the proposed example has *one machine* with *multiple modes of operation*, it is formulated in the same manner that a *multiple single-mode machines* example would. As such, in the following, a *mode* represents a *machine* in our problem formulation.

We studied the robustness and scalability of the proposed strategy on a series of variations in the characteristics of the tapping process. The model parameter sets were:

$$\begin{aligned} F_d &\in \{0.05, 0.1, 0.2, 0.5\} \qquad F_s \in \{0.01, 0.2, 0.5, 1\} \\ G_r^{min} &\in \{1, 2, 5, 10\} \qquad G_r^{range} \in \{2, 5, 10, 20\} \\ n &\in \{3, 4, 5\} \qquad J \in \{0.1\} \qquad V_a \in \{10, 20\} \end{aligned} \tag{13}$$

The parameters not listed were set as in Eq. (7). The Cartesian product of these variations generated a total of 1536 parameters configurations, 972 of which resulted in a machine's configuration that was unable to reach the target depth, and 564 valid configurations. We ran one experiment for each configuration, in which the modes differed only by gear ratios. Gear ratios were calculated by linearly interpolating between G_r^{min} and $G_r^{min} + G_r^{range}$. Consequently, of the 1536 experiments, there were 512 with 3, 4, and 5 modes each. The experiments were configured with $x_{ini} = \mathbf{0}$, and a target depth of $40\,\mathrm{mm}$. As described in Sec. IV-B, we ran *Heuristic* and *Exhaustive* with exponentially decreasing discretization time h, beginning at $51.2\,\mathrm{s}$, and halving each round until either reaching $0.2\,\mathrm{s}$, or exceeding a timeout of five seconds. All experiments were run on a Windows 10 desktop machine, using a single core of an Intel® i5-6600K processor running at ~3.50 GHz.

A. Results analysis

As an example, Fig. 4 shows the Pareto fronts generated by one experiment, configured with $F_d = 0.1$, $F_s = 0.2$, $V_a = 20$, $n = 3$, $G_r^{min} = 1$, and $G_r^{range} = 5$. To aid in presentation, only the smallest three discretization times h are shown, i.e., 0.2, 0.4, and 0.8 seconds.

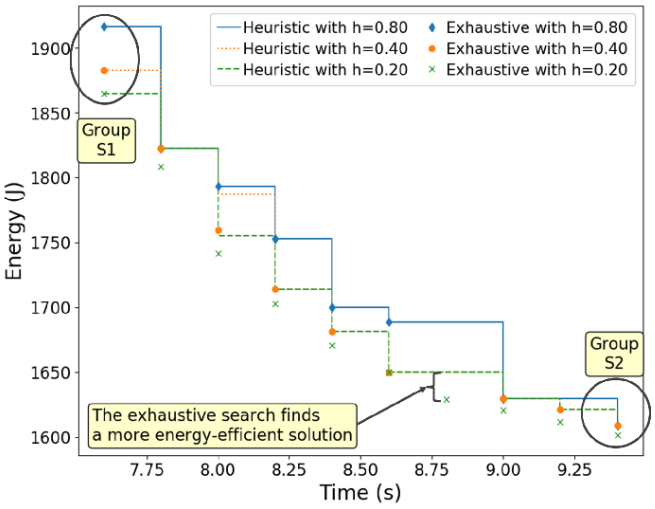

Fig. 4. Comparison between *Heuristic* and *Exhaustive* Pareto fronts for the smallest three discretization times h, i.e., 0.8, 0.4, 0.2.

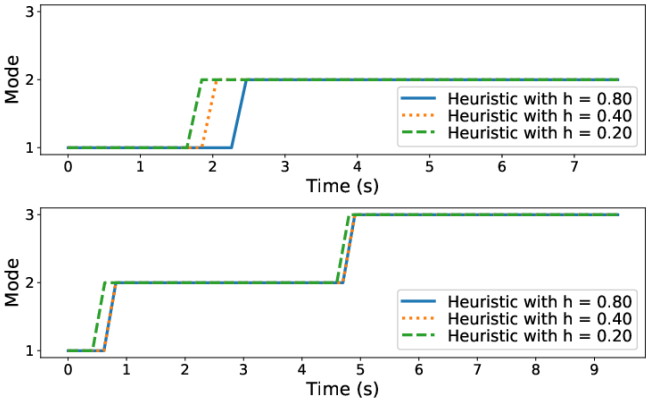

Fig. 5. Mode sequences for the Pareto optimal solutions found by the *Heuristic* minimizing processing time (top) or energy (bottom). Corresponding to *Group S1* and *Group S2* in Fig. 4 respectively.

As we can see in the figure, *Heuristic* and *Exhaustive* found equivalent (probably identical) solutions at discretization time 0.8 and most of the time also for 0.4. Only at a discretization time of 0.2 does the *Exhaustive* algorithm find almost consistently better solutions (crosses) in terms of Energy than the *Heuristic* (dashed line). Note however, that the differences are very small.

For reference, when disallowing mode switches during processing, the resulting Pareto front would have a single solution at $8.4\,\mathrm{s}$ and $1912\,\mathrm{J}$, corresponding to the exclusive use of mode 2.

The time and energy range covered by the Pareto-optimal multi-mode solutions, as well as the improvement in both dimensions compared to solutions without switching, demonstrate the advantages that mode scheduling can bring compared to treating each manufacturing job as an atomic operation in such a setup.

Now, let us focus on two groups of solutions on these Pareto fronts: the high-energy and low processing time *Group S1*, and the low-energy and high processing time *Group S2*. Specifically, let us focus on the three results that

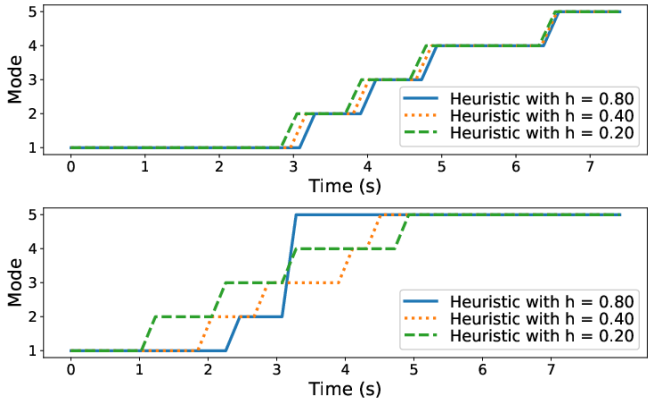

Fig. 6. Mode sequences for the Pareto-optimal solutions found by the *Heuristic* for another configuration. Again, the top shows the Pareto-optimal solutions that minimize low processing time and the bottom the ones minimizing energy.

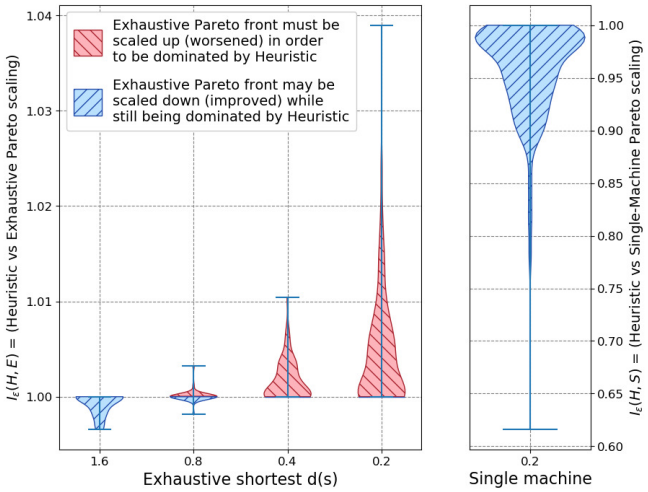

Fig. 7. ϵ-indicators comparing *Heuristic* with *Exhaustive* (left) and with a single machine and single mode (right). Values below 1 (blue color) imply that *Heuristic* dominates *Exhaustive*, whereas values over 1 (red color) indicate that at least some points in *Exhaustive* are not dominated by *Heuristic*. With $h = 0.2$, most of *Heuristic* Pareto fronts are less than 1% worse than those of the *Exhaustive*. *Heuristic* dominates the trivial single-machine solution.

the *Heuristic* is able to find for different values of h, i.e., 0.2, 0.4, and 0.8. Fig. 5 shows the mode switching behaviour for *S1* (top) and *S2* (bottom). In case of *S1*, the *Heuristic* decided to use only modes 1 and 2. Thanks to the lower granularity of the simulation, with $h = 0.2$ the switch from mode 1 to mode 2 occurs earlier than with $h = 0.8$. The total processing time is around $7.6\,\mathrm{s}$. For *S2* the difference between the switching instants is smaller, with a greater total processing time of around $9.4\,\mathrm{s}$.

In order to show that finer granularity levels not only lead to more finely tuned switching times, but can find completely different strategies, we also show the mode sequence for Pareto points found for another configuration in Fig. 6. This configuration has $n = 5$ modes and $F_d = 0.1$, $F_s = 0.01$, $V_a = 20$, $G_r^{min} = 2$, and $G_r^{range} = 2$. The solutions that minimize processing time (top) are again very similar between the different discretization times. However, the solutions that optimize for low energy (bottom) differ widely, and in the case of $h = 0.8$, mode 4 is skipped all together.

To aggregate the results of all our experiments, and to compare the *Heuristic* against the *Exhaustive* search, we use the *binary ϵ-indicator* [21], a metric that compares the Pareto fronts from two multi-objective optimization algorithms. Given two Pareto fronts Ω_1 and Ω_2, the binary ϵ-indicator is given by

$$I_\epsilon(\Omega_1, \Omega_2) = \max_{\omega_2 \in \Omega_2} \min_{\omega_1 \in \Omega_1} \max_{i \in \{1,2\}} \frac{\omega_1(i)}{\omega_2(i)}. \qquad (14)$$

Intuitively, the value $I_\epsilon(\Omega_1, \Omega_2)$ is the smallest factor by which Ω_2 may be scaled so that it is completely dominated by Ω_1. Values below 1 imply that Ω_1 already dominates Ω_2, whereas values over 1 indicate that at least some points in Ω_2 are not dominated by Ω_1.

The violin plots on the left of Fig. 7 show the values of $I_\epsilon(H, E)$ across all experiments performed, where H is the Pareto front from *Heuristic* with discretization time $0.2\,\mathrm{s}$, and E is the Pareto front from the last complete iteration of *Exhaustive*. The results are binned by that discretization time. As expected, if *Exhaustive* reaches the same granularity as *Heuristic*, its results are at least as good as those from *Heuristic*. In most cases (500 out of 564) however, the ϵ-indicator is less than 1.01, i.e., the Pareto front from *Heuristic* is at most $1\,\%$ worse than that from *Exhaustive*. Even in the worst case, ϵ is less than 1.04. When *Exhaustive* is not able to finish, the differences shrink, and in some cases *Heuristic* is even able to find better solutions. This demonstrates the effectiveness of the *Heuristic* in finding near-optimal solutions.

In order to provide a baseline, we also compared the Pareto fronts found with the *Heuristic* with those that result from selecting the mode only at the beginning, with no switches allowed during processing. As can be seen in the violin plot on the right of Fig. 7, *Heuristic* usually dominated these trivial solutions both in terms of time and energy, and sometimes significantly so.

B. Efficiency analysis

Fig. 8 shows the runtime of each of the algorithms for each experiment. The elapsed time is shown on the vertical axis in a symmetric logarithmic scale, and the experiments are shown across the horizontal axis, sorted by *Exhaustive*'s runtime. It can be seen that *Heuristic* completes significantly faster than *Exhaustive* in all cases, usually by one or more orders of magnitude. Indeed, the longest run took less than $150\,\mathrm{ms}$, and it took less than $4\,\mathrm{s}$ in total to run *Heuristic* for all experiments. Thus, although *Heuristic* may produce solutions that are slightly worse than the true Pareto front found by *Exhaustive*, the massive improvement in runtime may make this a desirable trade-off for flexible manufacturing.

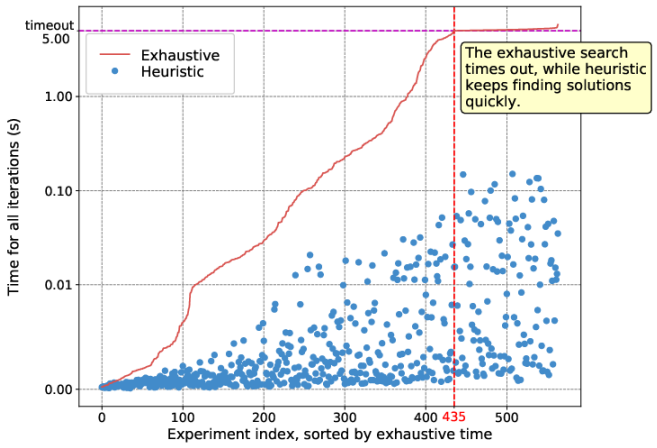

Fig. 8. Runtime comparison between heuristic and exhaustive searches. Heuristic is usually 1–2 orders of magnitude faster than the exhaustive search, and most importantly never timed out.

VI. Concluding Remarks

Analysis of hybrid systems and optimized scheduling of control tasks have been extensively studied in the context of cyber-physical systems. Yet, in smart manufacturing, individual processing steps have—to the best of our knowledge—only been treated as atomic work items. This is the first work to propose a holistic way to model computer-controlled manufacturing processes with hybrid dynamics. This enables manufacturers to optimize and explore possible trade-offs between different processing steps. We validated our modeling approach by developing an optimization strategy and applying it to a tapping process.

While initial results have been promising, it is important to further evaluate the potential of this approach. Towards that goal, we plan to apply our approach to processes with more complex (e.g., non-linear) system dynamics and a non-negligible cost for mode transitions, while also addressing additional real-world concerns like material degradation or tool aging. In this context, we will also improve our basic optimization strategy, like discretizing with dynamic step lengths. The problem we studied in this paper also has interesting connections to electronic design automation involving analog components [22], [23]. We plan to study how the solutions proposed for such problems might extend to our setup and vice versa.

VII. Acknowledgements

This work is supported by the US NSF grant #2038960. The authors also gratefully acknowledge the feedback of the anonymous reviewers that helped in improving the paper.

References

[1] Z. Cao, C. Lin, and M. Zhou, "A knowledge-based cuckoo search algorithm to schedule a flexible job shop with sequencing flexibility," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 56–69, 2021.

[2] L. Alting and H. Zhang, "Computer aided process planning: the state-of-the-art survey," *International Journal of Production Research*, vol. 27, no. 4, pp. 553–585, 1989.

[3] A. Nahhas, S. Lang, S. Bosse, and K. Turowski, "Toward adaptive manufacturing: Scheduling problems in the context of industry 4.0," in *6th International Conference on Enterprise Systems (ES)*. IEEE, 2018, pp. 1–8.

[4] 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 Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 3, pp. 530–541, 2015.

[5] M. Ghaleb, S. Taghipour, and H. Zolfagharinia, "Real-time optimization of maintenance and production scheduling for an industry 4.0-based manufacturing system," in *Annual Reliability and Maintainability Symposium (RAMS)*, 2020.

[6] W. Yang and S. Takakuwa, "Simulation-based dynamic shop floor scheduling for a flexible manufacturing system in the industry 4.0 environment," in *IEEE Winter Simulation Conference (WSC)*, 2017.

[7] S. Chakraborty and P. Ghosh, "Heat exchanger network synthesis: the possibility of randomization," *Chemical Engineering Journal*, vol. 72, no. 3, pp. 209–216, 1999.

[8] M. Lukasiewycz *et al.*, "System architecture and software design for electric vehicles," in *50th Design Automation Conference (DAC)*, 2013.

[9] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Trans. Control. Syst. Technol.*, vol. 22, no. 6, pp. 2337–2345, 2014.

[10] A. Masrur *et al.*, "VM-based real-time services for automotive control applications," in *16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2010.

[11] A. Savkin and J. Somlo, "Optimal distributed real-time scheduling of flexible manufacturing networks modeled as hybrid dynamical systems," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 3, pp. 597–609, 2009.

[12] Y. Luo, C. Lu, X. Li, L. Wang, and L. Gao, "Green job shop scheduling problem with machine at different speeds using a multi-objective grey wolf optimization algorithm*," in *IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 2019.

[13] J. Wang, M. Qian, L. Hu, S. Li, and Q. Chang, "Energy saving scheduling of a single machine system based on bi-objective particle swarm optimization*," in *IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 2019.

[14] D. Roy, S. Ghosh, Q. Zhu, M. Caccamo, and S. Chakraborty, "GoodSpread: Criticality-aware static scheduling of CPS with multi-QoS resources," in *IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[15] S. Malik and D. Kim, "A hybrid scheduling mechanism based on agent cooperation mechanism and fair emergency first in smart factory," *IEEE Access*, vol. 8, pp. 227 064–227 075, 2020.

[16] M. D. Natale, A. Sangiovanni-Vincentelli, and F. Balarin, "Task scheduling with RT constraints," in *37th Design Automation Conference (DAC)*, 2000.

[17] A. Azim and S. Fischmeister, "Efficient mode changes in multi-mode systems," in *34th International Conference on Computer Design (ICCD)*, 2016.

[18] F. Sagstetter, P. Waszecki, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, "Multischedule synthesis for variant management in automotive time-triggered systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 637–650, 2016.

[19] R. Alur, V. Forejt, S. Moarref, and A. Trivedi, "Schedulability of bounded-rate multimode systems," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 3, pp. 1–27, 2017.

[20] M. Ruderman, J. Krettek, F. Hoffmann, and T. Bertram, "Optimal state space control of DC motor," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 5796–5801, 2008.

[21] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[22] A. V. Karthik and J. Roychowdhury, "ABCD-L: Approximating continuous linear systems using boolean models," in *50th Design Automation Conference (DAC)*, 2013.

[23] A. V. Karthik, S. Ray, P. Nuzzo, A. Mishchenko, R. Brayton, and J. Roychowdhury, "ABCD-NL: Approximating continuous non-linear dynamical systems using purely boolean models for analog/mixed-signal verification," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.