Exploiting Process Dynamics in Multi-Stage Schedule Optimization for Flexible Manufacturing

Michael Balszun[1], Clara Hobbs[2], Enrico Fraccaroli[2], Debayan Roy[1], Samarjit Chakraborty[2]
[1]TU Munich, Germany, [2]UNC Chapel Hill, USA

***Abstract*—The core idea of flexible manufacturing is adapting to changes. In this domain, the machine is not confined to a single fixed type of process but can perform different jobs (e.g., cutting, drilling) in different ways (e.g., varying speed, tool, power consumption). This adaptability should be enabled by a detailed view of how the machines work. The idea is to perform machine scheduling by exploiting the dynamical models—expressed as differential equations—of manufacturing processes, i.e., both machines and production items. The main innovation in this paper is the ability to compute a machine's schedule where the state of the product does not linearly evolve in time but is determined by the set of differential equations instead. Finding the schedule is defined as a multi-objective optimization problem—manufacturers may seek a trade-off between processing time, energy consumption, and other cost functions. The proposed optimization is evaluated using accurate process models, exemplifying how it works and harnesses the expressiveness of differential equations.**

***Index Terms*—Flexible manufacturing, scheduling, multi-objective optimization, particle swarm optimization**

I. Introduction

Industry 4.0 is poised to bring massive upheaval in the field of manufacturing. Thanks to smart sensors and Internet of Things devices, all levels of the manufacturing process are now interconnected, sharing data between production machines and analytics systems. This increased flow of data enables a fine-grained view of machines in a manufacturing process. Thus, as requirements change, it is now possible to automatically and better *adapt* processes to address these changes. In many cases, these changes come from new jobs to perform or completion of old ones. The production can accommodate these changes by creating a *schedule* of the operations to be performed for each job. The literature is rich in papers about how to *schedule* operations, even in an optimal way.

Unfortunately, much of the prior work on scheduling considers each operation as atomic, without considering variations that may be possible while a machine is operating on a part. For instance, a machine may have several different modes in which it can perform a given step (see Figure 1), each with different characteristics. Also, different machines might be able to perform different *types* of manipulations, and there might be a precedence constraint on the order of the manipulations. Carefully determining which mode to use at what times could increase the efficiency of the manufacturing process. Thus, the lack of modeling richness is a liability for Industry 4.0, where far more data on manufacturing processes are available than in the past.

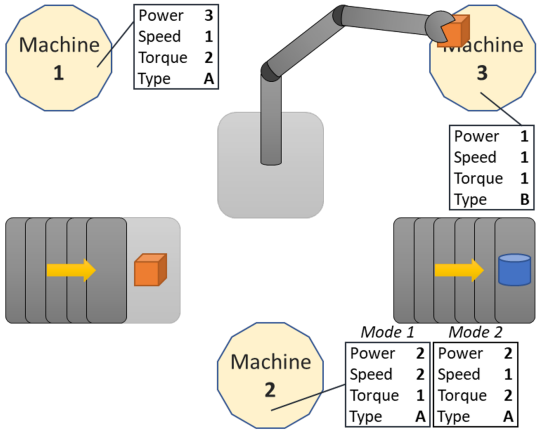

Figure 1. Flexible manufacturing multi-machine scenario

We propose a *cyber-physical systems* approach to manufacturing job scheduling to address this restriction. In this approach, we exploit the *dynamics* of the physical process, i.e., how the machine and the part evolve in an operational mode. Such physical dynamics can often be described mathematically using differential equations. This modeling is assumed to be done by domain experts and is outside the scope of this work. Given the mathematical models describing each mode, we intend to determine a time schedule for the machine to operate on the part in different modes.

To obtain this schedule, we formulate a *multi-objective optimization* problem—the factory may seek a trade-off between processing time, energy consumption, and other cost functions. In this work, we propose a *two-stage* approach to solve the problem. In the first stage, we *discretize* the physical dynamics according to a certain time granularity. Then, we perform a Breadth-First Search (BFS) to obtain the mode to be used in each discrete time step. We use an *iterative refinement* approach, progressively reducing the time step to determine the best time granularity, i.e., resulting in a reasonable search time. Overall, we derive a sequence of modes from the *first stage*, and a courser estimate of mode switching times. We further tune the switching times in the *second stage*, keeping the mode sequence unchanged and checking for better solutions. We have shown how Particle Swarm Optimization (PSO) can be applied for this fine-tuning.

A. Related work

Efficient machine scheduling is central to flexible manufacturing and has, therefore, attracted significant research atten-

tion in recent years. For instance, [1] discusses several scheduling techniques for Hybrid Flow Shop (HFS) problems, where a manufacturing process comprises a sequence of fixed jobs, with a set of identical machines capable of accomplishing each job. In such problems, a possible constraint is that the process cannot wait between two jobs, e.g., after a metal plate has been heated in a furnace, it shall be rolled without any delay. A machine schedule must respect such "no-wait" constraints while being deadlock-free [2] in cases when multiple processes use a pair of machines in different orders. Besides safety properties, machine breakdown and scheduled maintenance have been considered in such scheduling problems [3].

Towards modeling flexible manufacturing setups, a class of timed Petri nets has been investigated, where each job is modeled using a pair of "transitions" and a "place" [4]. Further, [5] formulates a weighted-sum multi-objective optimization model with precedence constraints between jobs. A real-time simulation model is presented in [6] that can assist in evaluating online decisions quickly in the event of a change (e.g., when high-priority jobs have arrived).

Social- and stochastic-based optimization techniques have been employed to solve the manufacturing job scheduling problem. In [7], for example, a multi-objective grey wolf optimizer is proposed to solve job shop scheduling problems while considering energy consumption and tardiness. Further, [8] uses a multi-objective PSO approach to determine the on-off times of a Computerized Numerical Control (CNC) machine. These approaches provide Pareto-optimal schedules, enabling a trade-off between energy consumption and production time.

All aforementioned works consider a job to take a fixed amount of time and may model the cost of switching between different types of tasks by a machine. However, unlike this paper, they do not consider that (i) a job can be carried out by a machine in different modes (equivalent to machines with different capabilities), (ii) the *process dynamics* evolve differently in each mode in continuous time, and (iii) the total time to accomplish a job will depend on the order of and the times spent on the selected modes. We note that [9] uses a hybrid model to capture flexible manufacturing systems. However, in that case, differential equations model how parts flow through the factory, while in this paper, we use them to model each job in a manufacturing process.

The scheduling problems in flexible manufacturing have mostly escaped the attention of the Real-Time Systems (RTSs) community, whose primary focus has been to study scheduling of software tasks and data frames on processors and communication buses, respectively. For example, [10] addresses how software tasks can be dynamically scheduled under real-time constraints and [11] considers multiprocessor scheduling of dynamic task graphs. The RTSs community has also investigated the interplay between task/message scheduling and physical dynamics in the context of cyber-physical systems [12]. Also, software task scheduling in a smart manufacturing setup has been considered in [13]. Furthermore, the RTSs community has investigated multi-mode system scheduling [14], [15]—where a processor switches between modes with different scheduling requirements—which has similarities to our problem setup. We note that in the aforementioned problems, only the order of tasks/messages must be determined and not their time duration, unlike in the problem under study.

We also point out that in control theory, switching between controllers has been considered, which leads to different continuous-time physical dynamics [16]. However, the main goal in such problems is to establish the stability of the system when arbitrary switching is allowed, while we are interested in determining switching instants so that production objectives are optimized.

Finally, we discuss a similar optimization problem in [17]. But instead of the two-stage approach as presented in this paper, [17] focuses on algorithmic improvements in the first stage. Additionally, in this paper we explore a more elaborate process model involving both switching times and cost

B. Contributions

This paper has the following main contributions:

- We call attention to a relevant yet unexplored scheduling problem in flexible manufacturing, namely, how a manufacturing step can be optimized by switching between different machine modes. To solve such a problem, we highlight that it is necessary to study the detailed process dynamics in each mode and the impact of switching between modes.
- A two-stage Pareto optimization technique is proposed. In the first stage, we use BFS to compute a sequence of machine modes and estimate time instants for switching between them. In the second step, we further fine-tune the switching instants using PSO.
- We evaluate our proposed approach for the well-known manufacturing step of drilling. Our experiments confirm that mode switching exposes significant optimization potential compared to a naïve single-mode solution.

C. Organization

The remainder of the paper is organized as follows. We first introduce our system model in Sec. II. Our two-stage optimization technique is detailed in Sec. III. Following this, we present an example of a process model in Sec. IV-A. We explain how the experiments are set up to evaluate the proposed technique in Sec. IV-B, and provide an analysis of the results in Sec. IV-C. Finally, Sec. V concludes the paper.

II. System modeling

Typically, in a manufacturing process, raw materials are presented as input, and are then processed by one or more machines to be transformed into finished products (or goods) as output. As this is a physical process, the state of the production item continuously changes in each manufacturing step. Also, this state change strongly depends on the dynamics of the machine being used for the process. At the same time, how the machine changes its state will depend on the state of the production item. Hence, the mathematical model of a

production process must be formulated by considering both the dynamics of the item and machine in tandem.

Let us denote a time-varying vector $x(t)$ as the *state* of the production process (i.e., the machine and the production item). For a manufacturing step using a single machine, we can write that the process state evolves according to the equation

$$\dot{x}(t) = \mathcal{A}(x(t)), \tag{1}$$

where $\mathcal{A}(\cdot)$ can be a linear or a nonlinear function of $x(t)$.

A typical production line consists of several machines to process materials, and a transport system. Not all machines can perform the same task, and among those that perform the same task, their characteristics may be different, e.g., energy usage or work speed. In our problem setting, we consider that a production process (or a part of the whole process) is equipped with a set of n machines, denoted $\mathcal{M} = \{m_1, \ldots, m_n\}$. When machine m_i is used to do the job, the process dynamics change according to a state-transition function $\mathcal{A}_i(\cdot)$, as in Eq. (1). The goal in this paper is to optimally select a machine (among the available ones) at each time t according to certain production objectives (e.g., minimize the production time and/or energy). Here, we denote the choice of machine m_i at time t using a function $\sigma(t) = i$. Hence, we can describe a manufacturing process (except during the transition between machines) as a switched dynamical system by combining Eq. (1) and the function $\sigma(t)$, leading to the form:

$$\dot{x}(t) = \mathcal{A}_{\sigma(t)}\left(x(t)\right). \tag{2}$$

We note here that transitions might not be possible between all machines in a realistic setup. This leads to a set of *allowed transitions* $\mathcal{P}$ between machines, each denoted by an ordered pair (i, j). Such a pair means that the system can switch from machine m_i to machine m_j.

As Figure 1 shows, the time to move an unfinished piece from one machine to another may be non-negligible. The switching might involve moving between machines, or staying on the same machine but changing the tool or mode of operation. Regardless the nature of the switch, this operation takes *time*, has a *cost*, and might have repercussions on the process *state*. Let us denote t_k^+ and t_{k+1}^- as the instants where the k-th mode (or machine) transition started and finished, respectively. The time for the k-th mode transition is, therefore, given by $\Delta t_k = t_{k+1}^- - t_k^+$. During this transition, the state changes according to the equation

$$\begin{aligned} x(t_{k+1}^-) &= x(t_k^+) + \mathcal{S}\big(x(t_k^+), \sigma(t_k^+), \sigma(t_{k+1}^-)\big) \\ &= x(t_k^+) + \mathcal{S}_{i,j}\big(x(t_k^+)\big) \\ &\text{s.t. } \big(i, j\big) \in \mathcal{P} \\ &\text{with } i = \sigma(t_k^+) \text{ and } j = \sigma(t_{k+1}^-) \end{aligned} \tag{3}$$

where $\mathcal{S}(\cdot)$ is a function of the current state $x(t_k^+)$, the current machine $\sigma(t_k^+)$, and the next machine $\sigma(t_{k+1}^-)$ to which we are going to switch. So the process state evolves in continuous-time, while the process control software can cause discrete transitions between dynamical laws by changing the operational machine. This naturally leads to the question of how to schedule such transitions optimally.

Our process starts from an initial state x_{ini}, and must reach a target region X_{tar} in the state space. In addition to manufacturing tolerances, X_{tar} is a region and not just a point because we usually only care about the exact value of some dimensions in the x vector (e.g., the dimensions representing properties of the item being manufactured) and less about others (e.g., the state of the machines). We can then combine Eq. (2) and Eq. (3) to obtain the following constraint:

$$\begin{aligned} x_{ini} + \sum_{k=1}^{K_s+1} \int_{t_k^-}^{t_k^+} \mathcal{A}_{\sigma(t)}\left(x(t)\right)\, dt \\ + \sum_{k=1}^{K_s} \mathcal{S}(x(t_k^+), \sigma(t_k^+), \sigma(t_{k+1}^-)) \in X_{tar}. \end{aligned} \tag{4}$$

Here, K_s is the total number of times machines are switched during the manufacturing process. Let us define $\sigma_k = \sigma(t)$ where $t_k^- \leq t \leq t_k^+$. The scheduling problem in hand, therefore, is to determine, for $1 \leq k \leq K_s + 1$, (i) the sequence of σ_k and (ii) the corresponding switching time instants (t_k^-, t_k^+). Note that $t_1^- = 0$ and $t_{K_s+1}^+ = T$, where T is the total production time, i.e., the time between the start and the completion of the process. One of our objectives is to minimize T.

In addition to time, there are usually other cost factors, such as the energy needed by the process. Similarly to Eq. (1), we model the cost evolution as a (potentially nonlinear) function of the state:

$$\dot{e}(t) = C(x(t)) \tag{5}$$

Note that this is not a differential equation itself. However, one could define a combined system $\tilde{\dot{x}} = [x\ c]^T$ and $\tilde{x} = \tilde{\mathcal{A}}(\tilde{x})$ that treats the cost as part of the system state. In the interest of clarity, we will treat cost and state separately in this paper. Whether it is more efficient to model state and cost separately or combined in the implementation depends on the specific structure of $\mathcal{A}$ and C.

Just as with the state, we do not model the cost evolution during a transition in a continuous manner but treat it as an atomic change represented by $\mathcal{E}(\cdot)$, i.e.,

$$\begin{aligned} e(t_{k+1}^-) &= e(t_k^+) + \mathcal{E}\big(x(t_k^+), \sigma(t_k^+), \sigma(t_{k+1}^-)\big) \\ &= e(t_k^+) + \mathcal{E}_{ij}\big(x(t_k^+)\big) \\ &\text{with } i = \sigma(t_k^+) \text{ and } j = \sigma(t_{k+1}^-). \end{aligned} \tag{6}$$

III. Proposed optimization strategy

We propose a two-phase optimization strategy—outlined in Alg. 1—as a solution to the presented problem. In the first phase, we discretize the model and perform an exhaustive search across all possible mode sequences (lines 3–7 in Alg. 1). In the second phase, we use Particle Swarm Optimization to further optimize the exact switching times (lines 8–12 in Alg. 1).

A. Stage 1: Discrete search

In the first phase, we model the processing as a sequence of processing steps of a fixed length h, with optional machine switches in between. Each solution can be represented as a sequence of machines $\mathcal{I} = [m_1, m_2, \ldots, m_n]$, where each m_k

Algorithm 1: High-level overview of the proposed two-stage optimization strategy

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

```
/* Initialization                                   */
```
$\omega_{triv} = \text{find\_trivial\_solution}()$
$h = \omega_{triv}.\text{time}$
```
/* Stage 1: Discrete search                         */
```
$\Omega_d = \{\omega_{triv}\}$
while $h \geq h_{min} \wedge$ *not Timeout* **do**
  $h = h/2$
  $\mathbf{M} = \text{c2d}(\mathcal{M}, h)$
  $\Omega_d = \text{discrete\_search}(\mathbf{M}, \Omega_d)$
```
/* Stage 2: Refinement                              */
```
$\Omega_{pso} = \{\}$
forall $\omega_{base} : \Omega_d$ **do**
  $\omega = \text{pso}(\omega_{base})$
  $\Omega_{pso} = \Omega_{pso} \cup \{\omega\}$
$\Omega = \text{ensure\_pareto\_property}(\Omega_{pso})$

corresponds to "applying" machine m_k for one full interval of length h, with machine switches in between. As long as $m_k = m_{k-1}$ (i.e., the machine stays the same), the state and cost change can be computed recursively as:

$$\begin{aligned} x_k &= F_{m_k}(x_{k-1}) \\ e_k &= E_{m_k}(x_{k-1}) + e_{k-1} \end{aligned} \tag{7}$$

If $m_i \neq m_{i-1}$ we also have to consider the effects that switching has on the system state and cost. Thus, we generalize Eq. (7) to:

$$\begin{aligned} xq &= \mathcal{S}_{m_{k-1},m_k}(x_{k-1}) + x_{k-1} \\ x_k &= F_{m_k}(xq) \\ e_k &= E_{m_k}(xq) + \mathcal{E}_{m_{k-1},m_k}(x_{k-1}) + e_{k-1} \end{aligned} \tag{8}$$

where $\mathcal{S}_{i,i}(x)$ and $\mathcal{E}_{i,i}(x)$ are always $\mathbf{0}$. For notational convenience, we will write this as:

$$\begin{aligned} x_k &= \phi_k(x_{k-1}) \\ e_k &= c_k(x_{k-1}) + e_{k-1} \end{aligned} \tag{9}$$

Starting with the given initial state $x_{ini} = x_0$, we now perform a BFS to find all the optimal sequences $\mathcal{I}$. Those sequences are part of what is called the Pareto-optimal set of solutions. Each such solution represents a trade-off between all the optimization dimensions, i.e., it might favor one dimension with a relative worsening of one or more of the others. Each node ω_a in the search tree consists of

1) the sequence of selected branches (selected machines) $\mathcal{I}_a = [m_1, m_2, \ldots, m_k]$ leading to the current node;
2) the state $x_a = \phi_k(\phi_{k-1}(\ldots(\phi_1(x_0))))$ we get after applying the sequence of state transformations;
3) the accumulated cost of those transitions $e_a = \sum_{i=0}^{n} c_i(x_i)$.

If $x_a \notin X_{tar}$, we have not yet found a complete solution, and we expand the node further. For that, we create a new node ω_b for each possible machine m_i that can be selected next (including the current one), with $\mathcal{I}_b = [\mathcal{I}_a \ m_i]$ and x_b and e_b computed using Eq. (9). Otherwise, we have found a complete solution. The node does not expand further because the monotonically increasing cost functions mean we will never find a better solution by expanding an existing node further. As the state and cost can be computed starting from a sequence $\mathcal{I}$, storing those values as part of the node is not strictly necessary. However, when we expand new nodes during the search, we need the previous state and cost; thus, it is more efficient to cache those values in the nodes.

We want to point out that for the final evaluation of a complete solution ω_a, we do not just use e_a directly, as we only calculate e_a in discrete steps. However, the system will have reached the target area sometime between the final step k_a and the previous one $k_a - 1$. Thus, e_a will overestimate the actual cost to reach X_{tar} with the given machine sequence. As there is usually no closed-form solution to calculate the exact time where the system reached the desired target state, we decided to rely on linear interpolation to estimate the exact costs. For that, we perform a linear interpolation of the state evolution between x_{k-1} and x_k, calculate the fraction of h the system spends in X_{tar} and subtract that fraction from the last cost increment.

1) Discretization: In the most general case, F and E can be implemented by numerically solving the differential equation Eq. (1). Various algorithms for this exist, most notably the family of Runge–Kutta methods (see [18]). However, for many real-world problems, the differential equations can be discretized at the start of the algorithm, such that evaluation of F and E only requires a few matrices multiplications and additions. For instance, with linear systems where $\mathcal{A}$ is defined as $\dot{x}(t) = \mathbf{A}_i x(t) + \mathbf{B}_i u_i$, F becomes:

$$\begin{aligned} x[k] &= \Phi_i x[k-1] + \Gamma_i u_i, \\ \Phi_i &= e^{\boldsymbol{A}_i h} \\ \Gamma_i &= \int_0^h e^{\boldsymbol{A}_i s} \boldsymbol{B}_i \, ds = A_i^{-1}(e^{A_i h} - I)B_i. \end{aligned} \tag{10}$$

Where Φ_i and Γ_i need only be computed once at startup. Discretizing common cost functions, which are often quadratic, can be more complex. However, while the state change in a particular step depends on the previous step, the cost increment does not depend on the accumulated cost so far. As a result, errors that are introduced due to using some easier-to-compute interpolation have a much smaller total effect.

2) Selecting h through iterative refinement: The depth of the search tree corresponds to the number of steps the system needs to reach the target region. Moreover, because—in the worst case—the number of nodes grows exponentially with the depth of the search tree, choosing an appropriate discretization time is of utmost importance for the efficiency of this first phase. The general trade-off here is that a shorter discretization time means (in the worst case exponentially) longer algorithm run-time T and memory requirement. Conversely, a longer discretization time might mean we entirely miss advantageous machine sequences as we do not consider a switch at the necessary time. Different heuristics can be applied depending

on the exact process when choosing a step length. For instance, h should be short enough to capture the system dynamics, and if we have an initial solution, we can also estimate the tree depth for a given h. However, instead of predicting and balancing accuracy vs run-time, we propose an iterative refinement approach as outlined in lines 4–7 of Alg. 1. Here, we first run the search with a high h and then repeat it with a consecutively finer h until one of the following termination criteria is fulfilled:

- h reaches a predefined minimum h_{min}
- T or memory requirement reaches an upper bound
- The algorithm converges, and new iterations do not bring significantly better solutions

Repeatedly running the algorithm might seem wasteful at first. However, because—in the worst case—the number of possibilities to explore grows exponentially with the inverse of the length of the discretization interval, the run-time of the previous runs can usually be neglected compared to the final run. Also, the solutions found in one run can be used to prune the search tree in the subsequent runs and can thus even be a net win.

3) Transition constraints: The system is not necessarily allowed to transition from one machine to any other. This may be because it is physically impossible, or because we already know from a previous analysis step that specific orders of machines will not be worth considering. Algorithmically, such constraints can be taken into account in one of two ways:

- when expanding new nodes, the algorithm will not consider certain machines at all;
- encoding them inside the switching cost by setting the cost of these switches to infinite; as a result, the new partial solution will be pruned in the next iteration because any existing solution strictly dominates it.

4) Stability concerns: We are essentially describing a *switched* system. In control theory, an essential concern in dealing with such systems has been establishing stability when the system can arbitrarily switch between different dynamics. However, we have a different scenario here. We determine a fixed switching sequence that the process will follow and explicitly track the process state over time for the sequence. Hence, if the process diverges from the target state or oscillates around any intermediate state then such sequences will be pruned from the design space during the search. Also, we note that if the process is not stable when switching between two machine modes—as identified using control-theoretic techniques like common quadratic Lyapunov functions [19]—we can consider a transition constraint by leveraging our rich modeling approach.

5) Optimizations: Depending on the specific manufacturing setup, various optimization opportunities may exist in the first stage. For instance, one may employ heuristics to prune partial solutions that are unlikely to result in a Pareto-optimal solution. Alternatively, one could try performing an A*-search instead of the simple BFS if a suitable function to calculate a lower bound on the remaining cost can be found. Also, any restrictions on the possible machine transitions can be highly beneficial for reducing the overall execution time. For instance, if the transition graph contains no (directed) cycles, the worst-case complexity of the search algorithm is no longer exponential, but only polynomial. In general, however, there will be many situations where the h cannot be sufficiently small to realize the full potential of this new problem formulation. Therefore, we propose to further optimize the results from this first stage with a second stage that focuses on optimizing the switching times while leaving the order of machines unaltered.

B. Stage 2: Refinement via PSO

For the second stage in of the schedule optimization, we no longer define a solution via the sequence of discrete steps $\mathcal{I}$, but via the length of each segment (i.e. the time between two switches) $\mathcal{T}_s = [st_1, st_2, st_3, \ldots]$. Each of these segment times, except the last one, maps to a dimension in the solution space, which can then be explored by a multidimensional optimization algorithm—in our case, Particle Swarm Optimization. The last segment time is then determined by how long the system needs to reach X_{tar}. Note that $\mathcal{T}_s$ is only meaningful when accompanied by the selected machine, but since we optimize each point of the Pareto front separately (lines 10–11 in Alg. 1), the machine sequence is the same for all solutions evaluated during the refinement process of an individual schedule. In future work, we plan to investigate the use of multi-objective PSO algorithms that could optimize the whole Pareto front simultaneously. The main challenge is that each element of the Pareto front may have a different number of switches and a different sequence of machines, requiring a different solution space than that used here.

We use a standard PSO algorithm [20], onto which we map our problem as follows:

- To evaluate the fitness of a particular solution, we simulate the system according to the equations described in Sec. II until the target region is reached.
- When comparing the fitness of two solutions, one solution A is considered better than another B if and only if e_B is worse in at least one dimension than e_A and at most as good in all others. Here we make use of the fact that PSO only requires partial ordering between the design points and not a scalar metric.
- To initialize the PSO, we uniformly distribute the particles around the initial solution from the discrete search and add one particle at the initial $\mathcal{T}_s$.
- For lower and upper bounds, we allow a deviation of $\pm 2h$ from the base solution by default. The expectation is that if a dominant solution exists outside that range, the discrete search has produced another Pareto point closer to it anyway. Clearly, this property is not guaranteed but is an effective heuristic to limit the search space. Additionally, $st_i \geq 0$ must be enforced.

After running the PSO-based refinement for each of the Pareto points, we finally combine the individually optimized solution to a new Pareto front which is the final result from our proposed optimization scheme. Note that this new Pareto front

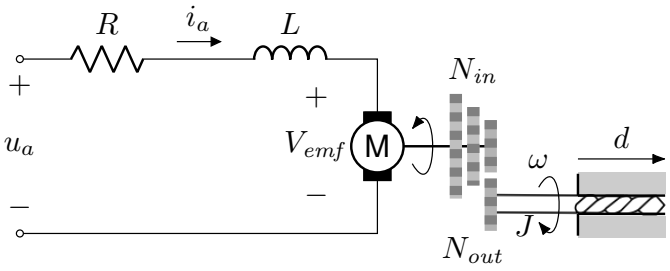

Figure 2. Schematic of the DC motor powering the drill.

may contain fewer solutions than after the discrete search, because an optimized solution might now dominate another one (line 12 in Alg. 1).

IV. Case study

This section is organized as follows. First, it introduces a case study. Then, it explains how the case study is used to evaluate the proposed methodology. Finally, it provides insights on the experimental results.

A. System Model

Let us now introduce an example process consisting of a CNC drilling machine with a Direct Current (DC) variable-speed motor that will be useful for evaluating our proposed technique. Speed variation is obtained by a set of gears and controlled by a computer. Changing the gear setting changes the output angular speed and torque, thereby changing the machine's dynamics.

A DC motor can be described at different levels of detail; here, we rely upon the differential equations governing the motor, assuming a homogeneous magnetic field [21]. These equations can be extended to encompass several aspects of the process being modeled. For instance, as the depth of the drilled section increases, so does the friction exerted on the drill bit. Alternatively, a prolonged processing time influences the motor temperature that impacts on specific motor parameters (e.g., winding resistance, the flux density of the permanent magnets). We consider the voltage applied to the armature winding as the input provided to the system.

The overall system is shown in Figure 2, and can be decomposed into an electro-mechanical and a rotational mechanical side. In this figure, everything on the left-hand side of the gears is the electro-mechanical side, while everything on the right-hand side of (and including) the gears is the rotational mechanical side. On the electro-mechanical side, resistance R models how the armature limits the motor's maximum current alongside the back-EMF voltage V_{emf}, while the inductance L models how it limits the current over time. On the rotational mechanical side, we have motor shaft connected to n input gears. Each input gear i has $n_{in}^{(i)}$ 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 drilling process dynamics are governed by the state-space model:

$$\dot{x}(t)_i = \begin{bmatrix} \frac{K_t}{J} & -\frac{K_d}{J} & -\frac{F_d\, G_r^{(i)}}{J} \\ -\frac{R}{L} & -\frac{K_e}{L} & 0 \\ 0 & \frac{T_s\, G_r^{(i)}}{2\pi} & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & -\frac{G_r^{(i)}}{J} \\ \frac{1}{L} & 0 \\ 0 & 0 \end{bmatrix} u, \quad (11)$$

where the state vector $x = [i_a\ \omega\ d]^T$ and the constant input vector $u = [v_a\ F_s]^T$, and 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 drill. The constants 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, respectively. The constant J gives the rotational inertia. The upper-right-most entry $F_d G_r^{(i)}/J$ in Eq. (11) accounts for the frictional force that depends on the drill'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 by using the element $T_s G_r^{(i)}/(2\pi)$ in Eq. (11).

We obtain different process dynamics for each gear ratio $G_r^{(i)}$. For n different input gears, the process can switch between n different dynamics. At time t, the process dynamics are given by $\mathcal{A}(x(t))_{\sigma(t)} = \dot{x}(t)_i$ where $\sigma(t) \in \{1, 2, \ldots, n\}$. As explained in Section II, our main objective is to find an optimal way to switch between these different dynamics.

We choose a two-dimensional metric for the switching cost where the first dimension corresponds to the processing time and the second to the used energy computed as the integral over $v_a \cdot i_a(t)$. For the switching of gears, we use fixed, state independent values:

$$\begin{aligned} \dot{C}(x) &= \begin{bmatrix} 0 & 0 & 0 \\ v_a & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ \mathcal{E}_{ij}(x) &= \begin{bmatrix} S_t & S_e \end{bmatrix}^T \end{aligned} \quad (12)$$

At the start of the discrete search phase, we discretized the model from Eq. (11) according to Eq. (10). Inserting the solution for $x(t)$ into the energy cost function $\dot{C}_e(x) = [v_a\ 0\ 0] \cdot x$ and integrating from 0 to h also yields an closed form solution for the discrete energy cost function for a processing step E:

$$\begin{aligned} E_e(x_k) &= x_{k-1}^T \mathcal{N}_h u + u^T \mathcal{R}_h u \\ \mathcal{N}_h &= \left(A^{-1}\left(e^{Ah} - I\right)\right)^T N \\ \text{with}\ \ \mathcal{R}_h &= \left(A^{-2}\left(e^{Ah} - I - Ah\right) B\right)^T N \\ N &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}^T \end{aligned} \quad (13)$$

For efficiency reasons, $E_e(x_k)$ does not include the time dimension of the cost which is trivially h. Also note that $\mathcal{N}_h u$ and $u^T \mathcal{R}_h u$ need only be computed once at startup, just as Φ_i and Γ_i in Eq. (10).

B. Experimental setup

We applied the two-stage optimization to the motivational example presented in Sec. IV-A. We implemented the whole methodological flow in C++.

Table I
PARAMETERS SWEEPS FOR THE DRILLING CASE STUDY

Parameter	Symbol	Values
Armature Voltage	v_a	$\{10, 20\}$
Dynamic Hole Friction	F_d	$\{0.05, 0.15, 0.5\}$
Static Hole Friction	F_s	$\{0.01, 0.20, 1\}$
Number of Gears	n	$\{3, 4, 5\}$
Minimum Gear Factor	G_{min}	$\{0.1, 0.5, 2\}$
Gear Range	G_{range}	$\{1, 4, 10\}$
Fixed Switch Duration	S_t	$\{0, 1, 5\}$
Fixed Switch Energy	S_e	$\{0, 100, 500\}$

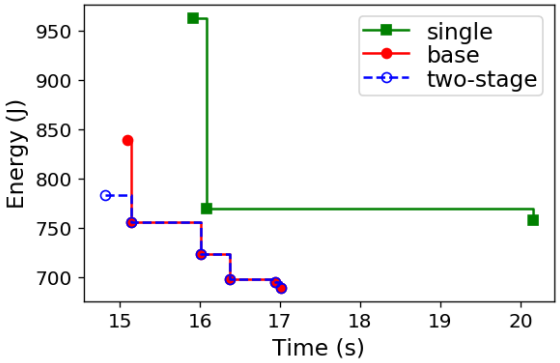

Figure 3. Pareto fronts from (i) using the same machine during the whole process (*single*), (ii) generated after the discrete search (*base*), and (iii) complete algorithm (*two-stage*)

Each parameter of the drilling process can influence the results and runtime of the proposed technique. In order to extensively evaluate its performance, we need to vary the process parameters. Another dimension to consider is the number of machines (or gear ratios) between which the process can switch. Each parameter configuration is an *experiment*, while a set of gear ratios are the number of *machines* available for that experiment identified by n.

We swept some of the process parameters through a range of values, reported in Table I. We varied the armature voltage and the two types of friction, i.e., dynamic and static. Further, we considered different combinations of gear ratios, specifically by changing the number of gears n to choose from, the minimum gear factor G_{min}, and the gear range G_{range}. Given these three variables, gear ratios were generated as n linearly spaced points between G_{min} and $G_{min} + G_{range}$. Finally, we considered different switching durations S_t and costs S_e. For our experiments, we kept them constant for all pairs of machines. The total number of combinations of parameters (i.e., *experiments*) resulting from this sweep is 4374.

C. Results investigation

We evaluate our proposed technique in different aspects. All comparisons are done in terms of *cost* and *time*. For the remainder of the section, we refer to the BFS algorithm as *base*, while we call *two-stage* the search algorithm plus the PSO. Finally, the naïve approach that uses the same machine is called *single*.

1) Pareto front: As an example, Fig. 3 shows the Pareto fronts at the different stages of the algorithm for a single experiment. The one labeled *single* shows all possible solutions if we only allow drilling with a fixed gear ratio. This roughly corresponds to the situation in classic job scheduling, where tasks are distributed to different machines as atomic jobs. Pareto fronts *base* and *two-stage* correspond to the solution sets Ω_d and Ω_{pso} from Alg. 1, respectively.

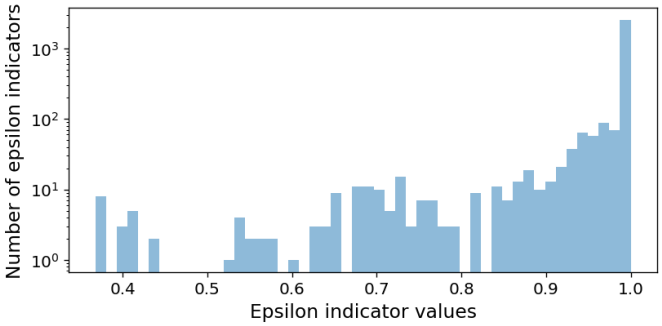

Figure 4. This histogram shows the distribution of the improvements achieved by the *two-stage* algorithm compared to the single machine solution.

As we can see, allowing mode switches has multiple advantages: (i) We can find solutions that perform the task faster and with less energy cost than possible when only using a single mode. (ii) We can extend the range of possible trade-offs as we get more intermediate solutions to choose from.

2) Comparison between single *and* two-stage *solutions:* Comparing Pareto fronts in an aggregated manner is difficult and many different metrics have been proposed. To give an idea of the overall potential, we compare the single machine Pareto fronts with the final Pareto front after the PSO using the *binary* ϵ*-indicator* [22]. Given two Pareto fronts Ω_1 and Ω_2, the binary ϵ-indicator is given by

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

Intuitively, the value $I_\epsilon(\Omega_1, \Omega_2)$ is the smallest factor by which Ω_2 may be scaled down, such that it is still completely dominated by Ω_1.

Fig. 4 shows in a histogram the distributions of the epsilon indicators. We use a logarithmic scale for the y axis of the histogram. There are about 2000 experiments where the epsilon indicator is in the neighbors of 1. However, a value of 1 does not mean that Ω_1 and Ω_2 are identical. Instead, the two fronts could share one or more points while all other points from Ω_2 are dominated by Ω_1. Similarly, a value greater than 1 would not necessarily mean that Ω_1 is worse than Ω_2. It means that at least one point in Ω_1 is dominated by Ω_2, while all others could still be non-dominated.

3) Comparison between base *vs* two-stage *solutions:* Finally, Fig. 5 shows the improvement that the *two-stage* algorithm could achieve over the *base* one. Here, we can perform a one-to-one comparison between the elements of Ω_d and Ω_{pso}. Individual Pareto points get often shared between the two and, hence, we do not use the epsilon indicators. Instead, we calculate how much the *two-stage* improves an individual Pareto point in terms of cost or time separately. Note the log scale of the histogram again.

In general, we see improvements that range from 0% to

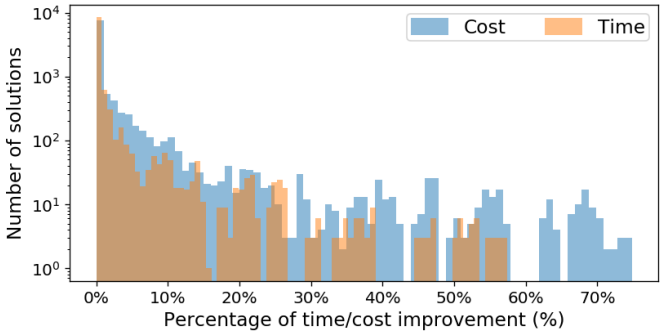

Figure 5. This histogram shows the distribution of the improvements achieved in the second stage compared to the first.

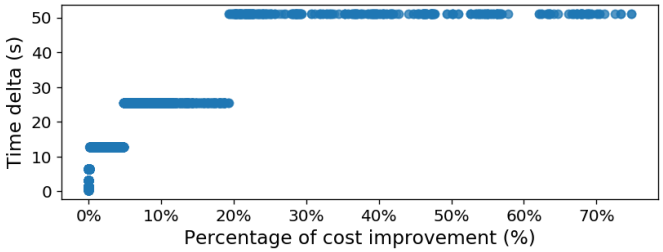

Figure 6. Scatter plot correlating the percentage of improvement achieved by the *two-stage* in terms of *cost* (x-axis) with the discretization step length (y-axis).

10%. However, in the best case we see improvements that are around 75% for *cost*, and 57% for *time*. Further analysis shows that these significant improvements correlate with the discrete search's high final h (see Fig. 6). This indicates that the PSO stage complements the first stage very well because it handles the pathological cases, where the exponential runtime of the first algorithm prevents a detailed search for the optimal switching point.

V. Concluding Remarks

This paper proposes a *cyber-physical systems* approach to manufacturing job scheduling, which models process dynamics by using differential equations. The machine scheduling is formulated as a *multi-objective optimization* problem which explores different trade-offs, e.g., processing time, energy consumption, and other cost functions. It combines in a two-stage optimization flow, a BFS to find which machines should be used and their order, followed by PSO to pinpoint the optimal hand-over time between them.

As next steps, we are particularly interested in applying our techniques to more complicated problems. Specifically, we want to integrate the results of this single-item scheduling with larger job-scheduling problems and investigate extensions or alternatives to PSO that optimize the selected machines and not only the segment durations.

References

[1] 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.

[2] 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.

[3] 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)*. IEEE, 2020.

[4] G. Cherif, E. Leclercq, and D. Lefebvre, "Modeling hybrid manufacturing systems using t-tpn with buffers," in *IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 480–487.

[5] K. Shen, J. David, T. De Pessemier, L. Martens, and W. Joseph, "An efficient genetic method for multi-objective continuous production scheduling in industrial internet of things," in *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1119–1126.

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

[7] 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)*. IEEE, 2019.

[8] 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)*. IEEE, 2019.

[9] 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.

[10] M. D. Natale, A. Sangiovanni-Vincentelli, and F. Balarin, "Task scheduling with RT constraints," in *37th Design Automation Conference (DAC)*. ACM, 2000, pp. 483–488.

[11] J. Huang, R. Li, X. Jiao, Y. Jiang, and W. Chang, "Dynamic DAG scheduling on multiprocessor systems: Reliability, energy, and makespan," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3336–3347, 2020.

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

[13] 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.

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

[15] 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.

[16] 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.

[17] M. Balszun, C. Hobbs, E. Fraccaroli, D. Roy, F. Fummi, and S. Chakraborty, "Process dynamics-aware flexible manufacturing for industry 4.0," in *IEEE 18th International Conference on Automation Science and Engineering (CASE)*, 2022, pp. 1–8.

[18] J. C. Butcher, *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[19] D. Cheng, L. Guo, and J. Huang, "On quadratic lyapunov functions," *IEEE Transactions on Automatic Control*, vol. 48, no. 5, pp. 885–890, 2003.

[20] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *IEEE International Conference on Evolutionary Computation Proceedings*. IEEE, 1998.

[21] 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.

[22] 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.