processes

MDPI

Article

A Rescheduling Strategy for Multipurpose Batch Processes with Processing Time Variation and Demand Uncertainty

Taicheng Zheng, Dan Li and Jie Li *

Centre for Process Integration, The Department of Chemical Engineering, The University of Manchester, Oxford Road, Manchester M13 9SS, UK; taicheng.zheng@manchester.ac.uk (T.Z.); dan.li@manchester.ac.uk (D.L.)
* Correspondence: jie.li-2@manchester.ac.uk

Abstract: In this paper, we address the problem of dynamic scheduling of a multipurpose batch process subject to two types of disturbances, namely, processing time variation and demand uncertainty. We propose a rescheduling strategy that combines several ideas. First, when we generate a new schedule, we simultaneously construct a Directed Acyclic Graph (DAG) to represent this new schedule. While each node in the DAG represents an operation, each arc represents the dependency of an operation on another. Based on this DAG, we then use a simple procedure to determine how long an operation is allowed to be delayed without affecting the current makespan. After that, when the new schedule is used for online execution, we trigger a rescheduling procedure only when (1) we infer from the predetermined delayable time information that the current makespan will be extended, or (2) we observe new demands, or (3) the current schedule is not guaranteed to be feasible. In the rescheduling procedure, only the affected operations are allowed to be revised, while those unaffected operations are fixed. By doing this, we can reduce system nervousness and improve computational efficiency. The computational results demonstrate that our method can achieve an order of magnitude of reduction in both the number of operation changes and the computational time with a slightly better long-term makespan, compared to the widely used periodically–completely rescheduling strategy.

Keywords: dynamic scheduling; multipurpose batch processes; rescheduling strategy; processing time variation; demand uncertainty

check for updates

Academic Editor: Yinlun Huang

Received: 11 November 2024
Revised: 11 January 2025
Accepted: 16 January 2025
Published: 23 January 2025

Citation: Zheng, T.; Li, D.; Li, J. A Rescheduling Strategy for Multipurpose Batch Processes with Processing Time Variation and Demand Uncertainty. *Processes* **2025**, *13*, 312. https://doi.org/10.3390/pr13020312

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Multipurpose batch processes have been widely used in process industries, such as chemical production [1–3], pharmaceuticals [4–6], and food processing [7,8] as these industries are subject to frequent changes in product demands and high product specialities. The use of multipurpose units (also called machines), such as multipurpose reactors, makes the production system adaptive to such dynamic and specialised environments.

Due to the dynamic nature of production activities, a multipurpose batch process will almost inevitably experience streaming uncertain events (also called *disturbances*) during the online execution of a schedule [9]. When disturbances are observed, if the scheduler does not revise the original schedule or re-generate a new schedule accordingly, it may worsen production objectives such as the profit or makespan. More seriously, the original schedule may become infeasible after adding the constraint of observed disturbances. Therefore, it is necessary to determine a *rescheduling strategy* that can be used to revise the current schedule to accommodate these disturbances. Essentially, a rescheduling strategy can be considered as a function whose input typically includes the current system states (such as

the operational status of machines and inventory levels of materials), the original schedule, and disturbances, and the output includes when- and how-to-reschedule decisions.

When designing a rescheduling strategy, we usually expect the strategy to have several desired properties. One of them is computational efficiency. In other words, we expect the strategy to respond quickly. For example, in an online production environment, the duration of an operation is often measured in a minute-level precision. In such settings, when a disturbance occurs, it is immediately detected by on-site sensors and uploaded to the Manufacturing Execution System (MES). Then, the Advanced Planning and Scheduling (APS) system, which appears as an integrated module in the MES, needs to process this disturbance information and decide whether to trigger a rescheduling process. If so, then the APS system is also required to return an updated schedule before the subsequent production activities are affected. Typically, all these steps need to be completed within a time window of several minutes. Therefore, computational efficiency is one of the highest priorities of designing a rescheduling strategy.

Another desired property is the ability to achieve a good long-term objective value, such as the long-term profit or long-term makespan [10,11]. Although it seems natural to expect this property, we would like to highlight that in practice, it is difficult to measure how good a long-term objective is. This is because achieving a good long-term objective in the context of dynamic scheduling is very different from obtaining a good objective value in a static scheduling problem. Specifically, for the static case, in most situations, we can model the problem as a Mixed-Integer Linear Programming (MILP) problem and use the MIP gap to measure how far the current objective is from the theoretical optimum. However, this approach does not apply in the dynamic situation. On the one hand, during the dynamic scheduling process, naïvely solving static scheduling problems as frequently as possible does not necessarily lead to a better long-term objective. On the other hand, the look-ahead information during dynamic scheduling is finite (meaning that we do not have access to the outcome of all future disturbances) and the dimensionality of state space is high (often hundreds to thousands). Therefore, it is unlikely to obtain, or even approximate, the optimal strategy, leading to difficulties in setting up a theoretical baseline for the long-term cost of a rescheduling strategy. Nevertheless, we still expect the long-term objective to be good in a comparative sense. That is, a good rescheduling strategy should achieve better, or at least not worse, long-term objectives when compared to the long-term objectives that are generated by well-known strategies, such as the periodically–completely rescheduling strategy.

The third desired property is that the rescheduling strategy should lead to a low level of system *nervousness* [12,13]. Here, the term "nervousness" refers to a measure of differences in schedules before and after rescheduling. The more different the original and new schedules are, or the more frequently the rescheduling process is triggered, the higher the level of system nervousness is. Although the influence of system nervousness does not directly reflect in the conventional objectives such as the cost and makespan, it does impact the actual production activity [13–15]. A high-level system nervousness may disrupt pre-processing procedures that are prior to the actual execution of a schedule, cause additional rearrangement costs, and even damage business credibility with third-party contractors [16]. Therefore, how to reduce system nervousness is also an important aspect when designing a rescheduling strategy.

To achieve these desired properties, the Process System Engineering (PSE) community has proposed numerous rescheduling strategies over the past decades. These strategies are often implemented by solving a series of MILP problems that are modified from static scheduling models. These modifications include modifying parameters [11,17], adding penalty terms [14,15], adding terminal constraints [18], partitioning the horizon [15,19],

and re-partitioning indices and sets [20,21]. The objectives of these modifications include (1) incorporating newly observed disturbances [22], (2) minimising the difference between the new schedule and the original schedule [15], (3) ensuring that some system states such as inventory levels are stable in the long term [11], and (4) generating some schedules that are more preferable than others [14]. Next, we review related state-of-the-art methods. After that, we highlight the contributions of this work.

Kanakamedala et al. [23] presented a reactive heuristic for schedule modification based on a decision tree. In the decision tree, each node represents a processing step (comparative to a task category, which is the term that is more commonly used today), and each arc represents a modification decision on that processing step. They considered two types of modification decisions, namely right-shifting and replacement (equivalent to reassignment). Since the size of the decision tree grows exponentially with the number of processing steps, they also proposed a two-stage method to prune the decision tree. Their work is different from ours because they considered only two types of modification decisions, including right-shifting and reassignment. In our method, it is also possible to modify the batching decision (how large a batch size to initiate) to obtain a better long-term objective. Also, their heuristic relies on a manually set parameter, "batch priority", which may not be trivial to provide in practice.

Elkamel and Mohindra [15] presented a reactive scheduling method that was intended to be used in a rolling-horizon pattern. They added penalty terms to minimise the difference between the new schedule and the original schedule. They considered four types of penalty terms, including task time-shifting penalties, assignment penalties, batch size preservation penalties, and resource purchase and reallocation penalties. In addition, they proposed an empirical rule to partition the scheduling horizon into three subhorizons, namely the left, the middle, and the right subhorizons. During the reactive scheduling procedure, only operations in the middle subhorizon were allowed to be revised, while those beyond the middle subhorizon were fixed.

Honkomp et al. [24] proposed a simulation-based framework to evaluate the robustness and expected performance of a particular rescheduling strategy. They divided a scheduling horizon into three subhorizons, namely a fixed scheduling subhorizon (where operations are not allowed to be modified), penalised scheduling subhorizon (where modifying operations will be penalised), and free scheduling subhorizon (where operations are allowed to be modified without being penalised). Although their work does not directly deal with a specific rescheduling strategy, it represents an important category of rescheduling strategies, that is, the simulation-based rescheduling strategy. When- and how-to-reschedule decisions are determined based on the results of discrete event simulation for a schedule. The strength of this category of methods is that the robustness of a rescheduling strategy can be proven using a statistical guarantee (given that enough episodes of simulations are performed). However, in practice, there may not be sufficient reactive time for carrying out these simulations when the computational efficiency is highly desired. In our method, we prioritise computational efficiency and choose a purely reactive strategy.

There are several works that deal with the reactive scheduling of multipurpose batch processes subject to machine breakdown and rush orders. Vin and Ierapetritou [22] used a two-stage approach to address the reactive scheduling of a multipurpose batch process. In the first stage, they used a baseline version of a continuous-time MILP formulation to generate a deterministic schedule. In the second stage, they modified the baseline MILP formulation by (1) fixing the variables before the time point of the occurrence of a disturbance event. This step was used to label the "already-executed" tasks. (2) For machine breakdown, they fixed the assignment decision while shifting subsequent operations on the

same machine and added penalty terms to the objective function to minimise the differences between the original schedule and the new schedule. (3) For rush order arrivals, they also fixed the assignment decisions for all operations in the original schedule, while adding constraints to model the additional demands. Janak et al. [19] contributed a subsequent work which also considered machine-breakdown and rush-order disturbances. They partitioned the scheduling horizon into the already-executed subhorizon and the currently executed subhorizon. For machine breakdown, they followed a strategy that depended on the specific machine that was broken down. That is, for unaffected machines, they fixed all tasks that were initiated before the shutdown event. For the affected machines, they fixed all the tasks that were completed before the shutdown event. For new orders, they used a set of heuristics to determine whether a task should be revised accordingly. The criteria for these heuristics included whether a task is executed on a "busy" machine and whether a task is executed on a "special" processing unit. Li and Ierapetritou [21] used a set of parameters to model the outcomes of disturbance events in the MILP formulation. They then used a multi-parametric programming approach to preventively (or, equivalently, proactively) generate a new schedule during dynamic scheduling. For machine breakdown, they followed the same strategy as the one used in [19]. For rush orders, they fixed tasks that were executed before the rush order came and updated the demand parameter after the rush order. These works are different from ours because (1) although they simulate the dynamic scheduling process by updating the values of a set of indicator variables (which indicates whether a task has been executed), they do not really follow the closed-loop pattern of a dynamic system. (2) While they generally follow a periodically rescheduling strategy, they do not explore the event-driven when-to-reschedule strategy. In our work, we will develop a dynamic system formulation to simulate the dynamic scheduling process at a full scale. Also, we will use an event-driven strategy to reduce long-term system nervousness.

There is a series of works, published after the 2010s, that model the dynamic scheduling process using a state-space model [11,14,17,25,26]. These works treat the MILP problem as a "schedule generator" and use the state-space model to describe the system dynamics. Generally, these works follow a periodically rescheduling strategy. As identified in [11], critical parameters that are involved in a periodically rescheduling strategy include the frequency of rescheduling, the length of the scheduling horizon, the computational resources that are allocated in a particular time step, how to deal with penalty terms, and how to add terminal constraints. To address the design of these parameters, McAllister et al. [14] discussed the theoretical aspects of various penalty terms under the framework of Economic Model Predictive Control (EMPC). They proposed a form of penalty terms called Hinge Loss of Discounted Difference, where they only penalised moving operations earlier and did not penalise delaying a task. Ikonen et al. [27] presented a reinforcement learning approach to determine the rescheduling timings and the computational resources that are required to be allocated in a specific rescheduling time step. In their work, the input of the reinforcement learning agent includes deviations in parameters in the MILP formulation, discrete changes (such as machine breakdowns), and the current state of the MILP solver. However, to the best of our knowledge, the warm start technique, which uses the information of the MILP solutions from previous time steps and therefore has the potential to apply dynamic scheduling to large-sized problems, has received limited attention from these works. In this work, we will also explore this technique.

As a result, the primary aspect of our contribution is that we propose an efficient rescheduling strategy that can deal with large-sized problems. As will be presented in Section 5, our method can handle the dynamic scheduling problem of a large-sized multi-purpose batch process within 1500 s. Importantly, we achieve this computational efficiency with a slightly better long-term objective (makespan, in our case) and significant reduction

in long-term system nervousness compared to the periodically–completely rescheduling strategy. The second aspect of our contribution is methodological. Our rescheduling strategy is a combination of several ideas. While some ideas may have been involved in previous works, some ideas are new. The fundamental idea of our strategy is using a Directed Acyclic Graph (DAG) to describe the inter-operation dependencies in a schedule. In the DAG, each node represents an operation, and each arc represents a dependency of an operation on another. With this DAG, we then propose a procedure to determine how long an operation is allowed to be delayed for without affecting the current makespan. To the best of our knowledge, the combination of these two ideas has not been proposed before. After that, these sets of information are used in our when-to-reschedule and how-to-reschedule strategies. Our when-to-reschedule strategy is essentially a hybrid strategy, which reschedules either at periodically distributed time points or when certain conditions are met. Our how-to-reschedule strategy uses the DAG and associated information of the maximum delayable time to enable a warm start throughout the entire dynamic scheduling process. We then test our method by solving a medium-sized problem and a large-sized problem. The computational results show that our method can achieve an order of magnitude of reduction in both the number of operation changes and the computational effort required for solving the MILP problems, compared to the existing periodic complete rescheduling strategy. More importantly, these improvements are achieved with a slightly better long-term makespan.

The remainder of this paper is organised as follows. In Section 2, we define the problem in detail. In Section 3, we present the detailed problem formulation, which describes the online execution of a dynamic scheduling process, and the MILP formulation for generating a new schedule. In Section 4, we propose the detailed rescheduling strategy, including the when-to-reschedule and how-to-reschedule strategies. In Section 5, we solve large-sized examples to illustrate the capability of our method. Finally, in Section 6, we draw conclusions and discuss future works.

2. Problem Description

We partition the problem description into two parts. The first part focuses on the static aspect of the problem, such as the basic components of a multipurpose batch process, basic assumptions of the system, and system parameters that are given a priori. The second part focuses on the dynamic aspect of the system, such as the pattern that the system evolves through time and the approach with which we model disturbances.

2.1. Static Aspect

We use the well-established State-Task Network (STN) to represent a multipurpose batch process. Figure 1 presents a sample STN, which is essentially a Directed Acyclic Graph (DAG). In the figure, each circle node represents a type of material. Each squared node (with solid border lines) represents a type of task. An arc that starts from one node and goes to another node represents a processing step. Typically, the manufacture of a product starts from a set of raw materials. These raw materials then follow a specific processing route in the STN and are finally converted to the desired products. During this process, materials may have different intermediate states (or equivalently, be converted into different intermediate materials). Each material is stored in a dedicated storage tank. Each tank has a finite storage capacity. Throughout this paper, we use indices and sets $k \in \mathbb{K}$ to denote materials (states). Also, we use $\mathbb{K}^{\mathrm{R}}$, $\mathbb{K}^{\mathrm{I}}$, and $\mathbb{K}^{\mathrm{P}}$ to represent the collection of raw materials, intermediate materials, and products, respectively. For the sample STN shown in Figure 1, there are the set of materials $\mathbb{K} = \{k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9\}$, the set

of raw materials $\mathbb{K}^{\mathrm{R}} = \{k_1, k_2, k_3\}$, the set of intermediate materials $\mathbb{K}^{\mathrm{I}} = \{k_4, k_5, k_6, k_7\}$, and the set of products $\mathbb{K}^{\mathrm{P}} = \{k_8, k_9\}$.

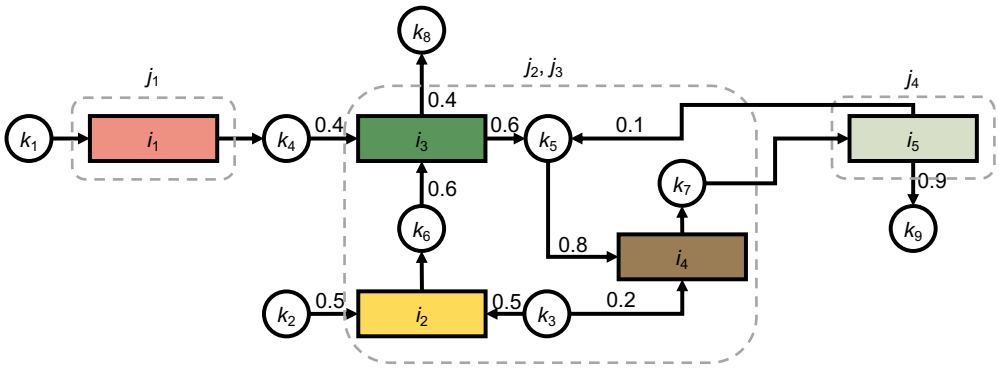


Figure 1. A sample State-Task Network (STN) that represents a multipurpose batch process.

In addition to tasks and materials presented in Figure 1, each round-corner box (with dashed lines) represents a set of machines (processing units). The inclusion relationships between dashed-line boxes (machines) and solid-line boxes (tasks) represent the compatibility between machines and tasks. That is, tasks in a dashed-line box represent the set of tasks that can be executed on this set of machines. Throughout this paper, we also use indices and sets $i \in \mathbb{I}$ to denote tasks, $j \in \mathbb{J}$ to denote machines, and $\mathbb{I}_j$ to denote the collection of tasks that can be executed on machine j. For the sample STN in Figure 1, there are the set of tasks $\mathbb{I} = \{i_1, i_2, i_3, i_4, i_5\}$ and the set of machines $\mathbb{J} = \{j_1, j_2, j_3, j_4\}$. According to the inclusion relationships denoted by dashed-line boxes, we have $\mathbb{I}_{j_1} = \{i_1\}$, $\mathbb{I}_{j_2} = \mathbb{I}_{j_3} = \{i_2, i_3, i_4\}$, and $\mathbb{I}_{j_4} = \{i_5\}$.

The decimal numbers next to an arc represent the conversion coefficient between tasks and materials. For example, in Figure 1, when initiating a batch of task i_3, the input materials should comprise 40% of material k_4 and 60% of material k_6, while the output materials consist of 40% of material k_8 and 60% of material k_5. We use ρ_{ik} to represent the conversion coefficient between task i and material k. If the coefficient ρ_{ik} is greater than 0, then task i produces material k. Otherwise, task i consumes material k. We also use $\mathbb{I}_k^+$ to denote the collection of tasks such that $\rho_{ik} > 0$ and $\mathbb{I}_k^-$ to denote the collection of tasks such that $\rho_{ik} < 0$. In other words, $\mathbb{I}_k^+$ and $\mathbb{I}_k^-$ represent tasks that produce and consume material k, respectively. We assume that the production process is invariant through time. That is, during the dynamic scheduling process, the STN structure and the conversion coefficients are time-invariant.

In addition to the conversion coefficients, the following system parameters are known a priori and are also time-invariant.

- $y_{ij}^{\min}$. The minimum batch size that is allowed to initiate task i on machine j.
- $y_{ij}^{\max}$. The maximum batch size that is allowed to initiate task i on machine j.
- τ_{ij}. The nominal processing time of executing task i on machine j. In other words, τ_{ij} represents the time that is required to execute task i on machine j without disturbances.
- $s_k^{\max}$. The storage capacity of material k.

Given the STN layout, the processing information, the supply-and-demand information (this will be mentioned in the next subsection), and the disturbance information (this will be mentioned in the next subsection), the target of scheduling is to determine a schedule that encodes four basic types of production decisions. That is, timing (when to initiate a batch), batching (how large a batch size to initiate), assignment (which machine to use to execute a batch), and sequencing (precedence relationships between batches). Besides these four basic types of decisions, we also consider the decisions of purchasing

raw materials and selling products. We assume that the supply of raw materials is sufficient compared to the demand of a product. In other words, we do not consider the case where the supply of raw materials is the bottleneck. Also, early delivery is allowed, meaning that the plant can fulfill an order earlier than its due date.

The objective is to minimise the *makespan*, which is defined as the total time required to complete all tasks that are necessary to fulfil all product demands.

2.2. Dynamic Aspect

We use a discrete-time dynamic system

$$\boldsymbol{s}_{t+1} = \textsc{TransFunc}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{i}_t), \quad t \in \{t_0, t_1, \cdots, t_e - 1\} \tag{1}$$

to describe the dynamics of a multipurpose batch process. Here, the index t represents discrete time points. We denote $\mathbb{T} = \{t_0, t_1, \cdots, t_e\}$ as the set of time points, where t_e represents the last time point. We call the duration $t_e - t_0$ the *timespan* of the dynamic scheduling process. In Equation (1), the vector of state variables $\boldsymbol{s}_t$ represents the physical states (e.g., machine status, inventory levels) of the system during half-open time interval $[t-1, t)$. The vector of action variables $\boldsymbol{a}_t$ represents the action that the system executes at time point t. The vector of information variables $\boldsymbol{i}_t$ represents disturbances during $[t-1, t)$. The transition function $\textsc{TransFunc}(\cdot)$ represents the transition rules of how system states evolve from time point t to $t+1$.

We model disturbances using random variables. We consider that the values of $\boldsymbol{i}_t$ are possible outcomes of a set of random variables $\mathbb{I}_t$. In our problem, we do not require that the probability distribution over information variables is known a priori. In other words, we do not assume the knowledge of $\mathsf{p}(\mathbb{I}_{t_0:t_e})$, where $\mathbb{I}_{t_0:t_e} = \bigcup_{t=t_0}^{t_e} \mathbb{I}_t$. However, we assume that the observed disturbance information is *perfect*. That is, once the values of information variables $\boldsymbol{i}_t$ are observed, their values are then confirmed and cannot change. Formally, let $\boldsymbol{i}_{t|t'}$ denote the vector of information variables during $[t-1, t)$, and particularly, the time point of observing this information is t'. Then by assuming the disturbance information is perfect, we have

$$\boldsymbol{i}_{t|t'} = \boldsymbol{i}_{t|t'+1} = \boldsymbol{i}_{t|t'+2} = \cdots,$$

which means that the observed values of the information variables $\boldsymbol{i}_t$ are invariant, no matter when they are observed (e.g., t', $t'+1$, or $t'+2, \ldots$). When the time point at which the scheduler observes the information is clear from the context, or is not the emphasis in the context, we can simply write $\boldsymbol{i}_{t|t'}$ as $\boldsymbol{i}_t$, which is consistent with the symbol used in Equation (1). It is noteworthy that although we assume that the disturbance information is perfect, our framework can be extended to the case where the observed value of a disturbance can vary from one time point to another. In this case, the varied disturbance can be modelled as the original disturbance plus a deviation disturbance. For example, when there is a rush order at time point t, the observed value of this rush order (i.e., the amount of this rush order) at t', $t'+1$, or $t'+2, \ldots$, can vary. However, the amount of this rush order observed at t', $t'+1$, or $t'+2, \ldots$, can be modelled as the invariant amount of the rush order plus a deviation.

As mentioned before, we assume that the system is subject to two types of disturbances, that is, the processing time variation and demand uncertainty. For the disturbance of processing time variation, we use a random variable V_{ijt} to model this type of disturbances. If there is a task i on machine j that initiates at time point t during the online execution, then its actual processing time will be delayed by v_{ijt} time periods. Here, the information variable v_{ijt} represents the realised value of V_{ijt}. The outcome of V_{ijt} is revealed to the

scheduler within a few time periods or just before the actual occurrence of the disturbance. Let h^V denote this time duration. Then for $\forall t \in \mathbb{T}$, the following information will be revealed to the scheduler during $[t-1, t)$,

$$v_{ij(t+1)}, v_{ij(t+2)}, \cdots, v_{ij(t+h^V)}.$$

For demand uncertainty, we consider two patterns of product demands. The first pattern is the *baseline* demand, which represents the periodic, regular demands from long-term contractors. We assume that the baseline demand occurs periodically and uniformly. We use μ_{kt} to denote the baseline demand of product k at time point t. For any $k \in \mathbb{K}^{\mathrm{P}}$ and $t \in \mathbb{T}$, the value of μ_{kt} is known to the scheduler a priori. The second pattern is the *intermittent* demand, which represents the non-periodic, irregular demands from casual customers. Rather than being known a priori, the intermittent demand is revealed to the scheduler gradually. We use u_{kt} and U_{kt} to denote the realised and unrealised intermittent demand for product k at time point t, respectively. Similarly, the following information will be revealed to the scheduler during $[t-1, t)$,

$$u_{k(t+1)}, u_{k(t+2)}, \cdots, u_{k(t+h^U)},$$

where we usually have $h^U > h^V$, which means that the irregular demand is more foreseeable than the event of processing time variation.

We use a sequence of vectors of action variables to represent a schedule under the formalism of a dynamic system. If at time point t' there is a schedule that encodes the values of action variables from t to $t + h_t$, we denote this schedule using

$$\boldsymbol{a}_{t:t+h_t|t'} = \left[\boldsymbol{a}_{t|t'}, \boldsymbol{a}_{t+1|t'}, \cdots, \boldsymbol{a}_{t+h_t|t'}\right], \tag{2}$$

where the reference time point t' is necessary, because a rescheduling process can change the planned action at the same time point. For example, it is possible that $\boldsymbol{a}_{t_5|t_2} \neq \boldsymbol{a}_{t_5|t_3}$, due to the rescheduling at the time point t_3.

The long-term objective of our problem is twofold. First, we aim to minimise the long-term makespan. We assume that both the baseline and intermittent demands can occur only during $[t_0, t_e^U)$, where t_e^U is a time point that is less than t_e. For example, we may set that $t_e = 1000$ and $t_e^U = 500$, which means that the demands can occur only in the first 500 time periods. Then, our objective is to design a rescheduling strategy to satisfy all the demands occurring during $[t_0, t_e^U)$ as soon as possible. The second objective is to minimise the long-term nervousness. We use d to denote the long-term nervousness, which is calculated by

$$d = \sum_{t=t_0}^{t_e-1} \textsc{NervFunc}(\boldsymbol{a}_{t:t+h_t|t}, \boldsymbol{a}_{t+1:t+h_{t+1}|t+1}), \tag{3}$$

where the nervousness function $\textsc{NervFunc}(\cdot)$ measures the schedule difference between time points t and $t+1$.

3. Problem Formulation

In this section, we present two detailed formulations for the dynamic scheduling problem. The first formulation is developed based on the dynamic system formalism. This formulation is used to model the dynamics during the online execution of a schedule for a multipurpose batch process. It describes how the system evolves through time in the physical world. The second formulation is developed based on the mathematical

programming formalism, which is essentially an MILP formulation. During the dynamic scheduling process, this MILP formulation is used as a "schedule generator". The inputs of the MILP formulation include the current system state, the original schedule, and the observed disturbance information, and the outputs include the optimal values of decision variables and the objective value. For clarity, we use lower case italic symbols (such as x) and overbar symbols (such as $\overline{x}$) to denote variables in the first formulation and second formulation, respectively.

3.1. Dynamic System

In this formulation, the following state, action, and information variables are defined.

3.1.1. State Variables

- r_{ijt}. An integer variable that represents the elapsed time of a batch of task i on machine j at the end of time interval $[t-1, t)$. If there is no such batch of task i that is running on machine j at the end of $[t-1, t)$, then $r_{ijt} = 0$. Otherwise, there is a batch of task i running, or there is a batch of task i about to complete, and then the value of r_{ijt} equals the elapsed time of this batch.
- b_{ijt}. A positive continuous variable that represents the current batch size of task i being processed on machine j. If there is no batch of task i that is running on machine j at the end of $[t-1, t)$, then $b_{ijt} = 0$. Otherwise, there is a batch running, or there is a batch about to complete, and then the value of b_{ijt} equals the current batch size of this batch.
- s_{kt}. A positive continuous variable that represents the inventory level of material k during time interval $[t-1, t)$.
- l_{kt}. A continuous variable that represents the backlog level of product material k during time interval $[t-1, t)$. In other words, l_{kt} represents the remaining unmet demand of product material k. Note that the value of l_{kt} can be less than 0 as early delivery is allowed.

3.1.2. Action Variables

- x_{ijt}. A binary variable that represents whether to initiate a batch of task i on machine j at time point t. If the batch is initiated, then $x_{ijt} = 1$. Otherwise, there is no batch initiated, and then $x_{ijt} = 0$.
- y_{ijt}. A positive continuous variable that represents the input batch size for initiating a batch of task i on machine j at time point t. If the batch is initiated, then the value of y_{ijt} equals its input batch size. Otherwise, there is no batch initiated, and then $y_{ijt} = 0$.
- p_{kt}. A positive continuous variable that represents the amount of raw material, k, that is purchased at time point t.
- q_{kt}. A positive continuous variable that represents the amount of production material, k, sold at time point t.

3.1.3. Information Variables

- v_{ijnt}. A positive integer variable that represents the processing time variation of a batch of task i. If there is a batch of task i on machine j, whose runtime is or will be n time periods by the end of $[t-1, t)$, then the completion time of this batch is delayed by v_{ijnt} time periods.
- u_{kt}. A positive continuous variable that represents the intermittent demand for product material k at time point t. The value of u_{kt} equals the demand for product material k at time point t.

3.1.4. Transition Function

Although the state-space model is obtaining increasing popularity in modelling the transition function of state variables, it is more straightforward to model a transition function using a piecewise function in our case. In our method, the value of a state variable is determined by a set of relational conditions. Our method does not require the tedious design of the transition matrix, and it is more intuitive to interpret.

For convenience, we also follow the convention that the logical operators (such as $\wedge$ and $\vee$) have lower operational precedence than the arithmetic operators (such as $+, -, \cdot$) and relational operators (such as $=, <, >$). For example, the expression $a = b \wedge c > d$ should be interpreted as $(a = b) \wedge (c > d)$ rather than $a = (b \wedge c) > d$. The transition functions for the state variables are given as follows.

$$r_{ij(t+1)} = \begin{cases} 0, & \text{if } \underbrace{(r_{ijt} = 0 \wedge x_{ijt} = 0)}_{(i)} \vee \underbrace{(r_{ijt} = \tau_{ij} + v_{ij(r_{ijt})t} \wedge x_{ijt} = 0)}_{(ii)}. \\ 1, & \text{if } \underbrace{(r_{ijt} = 0 \wedge x_{ijt} = 1)}_{(iii)} \vee \underbrace{(r_{ijt} = \tau_{ij} + v_{ij(r_{ijt})t} \wedge x_{ijt} = 1)}_{(iv)}. \\ r_{ijt} + 1, & \text{if } \underbrace{(r_{ijt} > 0 \wedge r_{ijt} < \tau_{ij} + v_{ij(r_{ijt})t})}_{(v)}. \end{cases} \tag{4}$$

Equation (4) represents the transition function for r_{ijt}. Here, the condition (i) represents the situation where there is no batch running or batch initiating at the end of $[t-1, t)$. As a result, we should have $r_{ij(t+1)} = 0$. The condition (ii) represents another situation where $r_{ij(t+1)} = 0$. The notable formula $\tau_{ij} + v_{ij(r_{ijt})t}$ represents the processing time after variation, where the first term, τ_{ij}, as mentioned before, represents the nominal processing time of the operation (i, j). The second term $v_{ij(r_{ijt})t}$ represents the delay time of this operation (recall the definition of v_{ijnt}). For cases where $r_{ij(t+1)} = 1$, the condition (iii) represents the situation that machine j is idle during $[t-1, t)$ but a batch is initiated at time point t. As a result, by the end of $[t, t+1)$, we should have $r_{ij(t+1)} = 1$. Note that, even if the varying processing time of this batch equals one time period, we still have $r_{ij(t+1)} = 1$, because this batch is about to complete by the end of $[t, t+1)$ (recall the definition of r_{ijt}). The condition (iv) represents a situation that is similar to the situation in (ii). The difference is that there is a batch initiated at the time point t. Finally, the condition (v) represents the situation where a batch is running during $[t-1, t)$ but is not completed by the end of $[t-1, t)$. As a result, the elapsed time should increase by one time period compared to the previous time point.

$$b_{ij(t+1)} = \begin{cases} 0, & \text{if } \underbrace{(r_{ijt} = 0 \wedge x_{ijt} = 0)}_{(i)} \vee \underbrace{(r_{ijt} = \tau_{ij} + v_{ij(r_{ijt})t} \wedge x_{ijt} = 0)}_{(ii)}. \\ y_{ijt}, & \text{if } \underbrace{(r_{ijt} = 0 \wedge x_{ijt} = 1)}_{(iii)} \vee \underbrace{(r_{ijt} = \tau_{ij} + v_{ij(r_{ijt})t} \wedge x_{ijt} = 1)}_{(iv)}. \\ b_{ijt}, & \text{if } \underbrace{(r_{ijt} > 0 \wedge r_{ijt} < \tau_{ij} + v_{ij(r_{ijt})t})}_{(v)}. \end{cases} \tag{5}$$

Equation (5) represents the transition function for b_{ijt}. The overall structure of this function is similar to Equation (4). For example, in the conditions (i) and (ii), when there are no new batches initiated, the current batch size of the batch should be 0. On the other hand, if there is a batch initiated at time point t, as presented in conditions (iii) and (iv), then the batch size by the end of $[t, t+1)$ should equal the input batch size y_{ijt}. Finally, as

presented in the condition (v), when the batch is running but not completing, the current batch size should equal the batch size at the previous time point.

To present the transition function for s_{kt}, that is, the inventory level of material k during $[t-1, t)$, it is helpful to introduce an intermediate variable, b^+_{ijt}, which represents the batch size that is output by the operation (i, j) by the end of $[t-1, t)$. Then, b^+_{ijt} is calculated by

$$b^+_{ijt} = \begin{cases} 0, & \text{if } (r_{ijt} < \tau_{ij} + v_{ij(r_{ijt})t}). \\ b_{ijt}, & \text{if } (r_{ijt} = \tau_{ij} + v_{ij(r_{ijt})t}). \end{cases} \tag{6a}$$

Here, the first condition represents the situation where the current batch is neither initiated nor completed. When the batch is completed by the end of $[t-1, t)$, which is represented in the second condition in Equation (6a), the output batch size equals b_{ijt}.

Then, the transition function for s_{kt} can be represented as

$$s_{k(t+1)} = s_{kt} + \underbrace{\sum_{j\in\mathbb{J}} \sum_{i\in\mathbb{I}^+_k \cap \mathbb{I}_j} \rho_{ik} b^+_{ijt}}_{(i)} + \underbrace{\sum_{j\in\mathbb{J}} \sum_{i\in\mathbb{I}^-_k \cap \mathbb{I}_j} \rho_{ik} y_{ijt}}_{(ii)} + p_{kt} - q_{kt}. \tag{6b}$$

Here, the term (i) represents the quantity of material that is added to the inventory of k. The term (ii) represents the quantity of material that is subtracted from the inventory of k. We define $p_{kt} = 0$ if $k \notin \mathbb{K}^{\mathrm{R}}$ and $q_{kt} = 0$ if $k \notin \mathbb{K}^{\mathrm{P}}$.

Finally, the transition function for the backlog level is simply a linear function, as presented in Equation (7).

$$l_{k(t+1)} = l_{kt} + \mu_{kt} + u_{kt} - q_{kt} \tag{7}$$

Here, μ_{kt} represents the baseline demand and u_{kt} represents the intermittent demand.

3.1.5. Nervousness Function

As presented in Equation (3), we need to measure the cumulative system nervousness using NervFunc$(\cdot)$. In this work, we define the nervousness function as below.

$$\begin{aligned} &\textsc{NervFunc}(\boldsymbol{a}_{t:t+h_t|t}, \boldsymbol{a}_{t+1:t+1+h_{t+1}|t+1}) \\ &= \sum_{t'=t}^{t+1+h_t} \sum_{j\in\mathbb{J}} \sum_{i\in\mathbb{I}_j} \left| x_{ijt'|t} - x_{ijt'|t+1} \right| \end{aligned} \tag{8}$$

To explain, let us discuss this formula case by case. Assume that we are examining a specific batch of task i on machine j at time point t' (typically $t' > t$). We compare the schedule of the version at the time point t (which we call the "original" schedule, represented by $\boldsymbol{a}_{t:t+h_t|t}$) and the schedule of the version at time point $t+1$ (which we call the "new" schedule, represented by $\boldsymbol{a}_{t+1:t+1+h_{t+1}|t+1}$). If, in both schedules, this batch does not exist, then we have $\left| x_{ijt'|t} - x_{ijt'|t+1} \right| = 0$. On the other hand, if, in both schedules, this batch is initiated, then we also have $\left| x_{ijt'|t} - x_{ijt'|t+1} \right| = 0$. However, in other cases where (1) the original schedule initiates this batch but the new schedule does not initiate it or (2) the original schedule does not initiate a batch but the new schedule initiates it, then we have $\left| x_{ijt'|t} - x_{ijt'|t+1} \right| = 1$. Therefore, Equation (8) can be intuitively understood as the number of batches that appear in the original schedule but not in the new schedule, added by the number of batches that do not appear in the original schedule but appear in the new schedule. It is noteworthy to mention that, even if we refer to these two versions of the schedule using the terms "original" and "new", it does not necessarily imply that a

rescheduling process is triggered at the time point t. At time points where the rescheduling process is not triggered, the value of NERVFUNC$(\cdot)$ should be equal to 0.

Finally, it should be noted that we only use Equation (8) as a metric for system performance, rather than as a threshold for determining whether to reschedule. This is because using Equation (8) as a threshold implies that a candidate schedule is required to be generated before making the whether-to-reschedule decision, which greatly increases the computational cost. As a result, we instead use a set of heuristics to check the condition for rescheduling, which is presented in Section 4.5.

3.2. Mixed-Integer Linear Programming Formulation

In this subsection, we present the MILP formulation that is used to generate a new schedule at time point t. For readers who are familiar with discrete-time formulations for scheduling of multipurpose batch processes, this MILP formulation can be viewed as a modified version of one of the standard discrete-time formulations proposed in [28,29].

As mentioned before, the inputs of this MILP formulation include the current system state at the time point t, the original schedule, and the observed information about disturbances. In our formulation, these inputs are incorporated either in the form of predetermined parameters or in the form of fixing the value of a decision variable. To distinguish the symbol t, which is used to denote time points in the dynamic system formulation, we use t' to represent the grid points that live within the MILP formulation. Also, recall that h_t represents the scheduling horizon, that is, the duration of the schedule that we generate at the time point t. Recall also that h^V represents the foreseeable horizon for processing time variation and h^U represents the foreseeable horizon for intermittent demands.

A notable parameter that we use throughout this subsection is $\tau_{ijnt'}$, which represents the varying processing time of a batch. If there is a batch of task i on machine j such that by the end of $[t'-1, t')$, its elapsed time is or will be n time periods, then the final processing time of this batch equals to $\tau_{ijnt'}$ time periods. Given the values of $v_{ijt}, \cdots, v_{ij(t+h^V)}$, we determine the value of $\tau_{ijnt'}$ as follows. For cases where $t' \leqslant t + h^V$, that is, within the foreseeable horizon, we set $\tau_{ijnt'} = \tau_{ij} + v_{ij(t'-n)}$. For cases where $t' > t + h^V$, we simply set $\tau_{ijnt'} = \tau_{ij}$, which is the nominal processing time.

3.2.1. Decision Variables

- $\bar{r}_{ijnt}$. A binary variable that represents the elapsed time of a batch of task i on machine j. If there is a batch of task i on machine j by the end of time interval $[t-1, t)$, and the elapsed time of this batch is equal to n time periods, then $\bar{r}_{ijnt} = 1$. Otherwise, $\bar{r}_{ijnt} = 0$.
- $\bar{b}_{ijnt}$. A positive continuous variable that represents the current batch size of a batch. If there is a batch of task i on machine j by the end of the time interval $[t-1, t)$, and the elapsed time of this batch is equal to n time periods, then $\bar{b}_{ijnt}$ equals its batch size. Otherwise, $\bar{b}_{ijnt} = 0$.
- $\bar{s}_{kt}$. A positive continuous variable that represents the inventory level of material k during time interval $[t-1, t)$.
- $\bar{m}_{kt}$. A positive continuous variable that represents the remaining unmet demand of product k during time interval $[t-1, t)$.
- $\bar{x}_{ijt}$. A binary variable that indicates whether a batch of task i on machine j is initiated at time point t.
- $\bar{y}_{ijt}$. A positive continuous variable that indicates the input batch size of a batch of task i on machine j at time point t.
- $\bar{p}_{kt}$. A positive continuous variable that represents the quantity of raw material k purchased at time point t.

- $\overline{q}_{kt}$. A positive continuous variable that represents the quantity of product k sold at time point t.
- $\overline{z}_t$. A binary indicator variable that represents whether there exist unmet demands during time interval $[t-1, t)$. If there are demands that are still unmet during $[t-1, t)$, then $z_t = 1$. Otherwise, all the observed demands are met during $[t-1, t)$, then $z_t = 0$.
- $\overline{ms}$. A positive integer variable that represents the makespan of the schedule.

3.2.2. Constraints

Initial state constraints. When establishing an MILP formulation that spans from time point t to $t + h_t$, we need to ensure that the values of the decision variables at the time point $t' = t$ are consistent with the system states at time point t. In our formulation, this consistency is achieved by adding a set of initial state constraints. As a result, there are four groups of decision variables that require fixing, namely $\overline{r}_{ijnt}$ (representing the elapsed time), $\overline{b}_{ijnt}$ (representing the batch size), $\overline{s}_{kt}$ (denoting the inventory level), and $\overline{m}_{kt}$ (denoting the unmet demand). The initial state constraints for these four groups of variables are presented in Equations (9)–(12), respectively.

Equation (9) represents the initial state constraints for $\overline{r}_{ijnt'}$.

$$M \cdot (\overline{r}_{ij(r_{ijt})t'} - 1) \leqslant r_{ijt} \leqslant M \cdot \overline{r}_{ij(r_{ijt})t'}, \quad \forall j \in \mathbb{J}, i \in \mathbb{I}_j, t' = t. \tag{9}$$

Here, M represents a large constant (e.g., 100). To explain Equation (9), let us examine it case by case. Assume that for a specific batch of task i on machine j, by the end of time interval $[t-1, t)$, the elapsed time of this batch is r_{ijt} time periods (greater than 0). Then, Equation (9) enforces that $\overline{r}_{ij(r_{ijt})t'} = 1$, which is consistent with the definition of $\overline{r}_{ijnt'}$. For cases where $r_{ijt} \neq n$ or $r_{ijt} = 0$, Equation (9) ensures that $\overline{r}_{ij(r_{ijt})t'} = 0$. As a result, Equation (9) ensures that the values of $\overline{r}_{ijnt'}$ variables at the time point $t' = t$ represent the current system states.

Similarly, Equation (10) encodes the batch size information at the time point $t' = t$.

$$M \cdot (\overline{b}_{ij(r_{ijt})t'} - 1) \leqslant r_{ijt} \leqslant M \cdot \overline{b}_{ij(r_{ijt})t'}, \quad \forall j \in \mathbb{J}, i \in \mathbb{I}_j, t' = t. \tag{10}$$

Equation (11) represents the initial state constraints for the inventory level. Because the definitions of $\overline{s}_{kt'}$ and $s_{kt'}$ are basically the same, we simply set their values as equal.

$$\overline{s}_{kt'} = s_{kt'}, \quad \forall k \in \mathbb{K}, t' = t. \tag{11}$$

Equation (12) represents the initial state constraints for $\overline{m}_{kt'}$, which represents the unmet demand within the scheduling horizon. As mentioned before, in this MILP formulation, we consider the total demand as the addition of baseline demands (μ_{kt}) and intermittent demands (u_{kt}).

$$\overline{m}_{kt'} = \sum_{t''=t}^{t+h^U} \mu_{kt''} + \sum_{t''=t}^{t+h_t} u_{kt''}, \quad \forall k \in \mathbb{K}^{\mathrm{P}}, t' = t. \tag{12}$$

Lifting constraints for elapsed time. Equation (13) represents the lifting constraints, where $\overline{r}_{ijnt}$ is the lifting variable that represents the elapsed time of a batch. Here, the term "lifting" means that these variables (such as $\overline{r}_{ijnt'}$ and $\overline{b}_{ijnt'}$) are lifted to a higher dimensional space (compared to r_{ijt} and b_{ijt}, for example). Intuitively, Equation (13) states that if the elapsed time of a batch by the end of $[t'-1, t')$ equals $n-1$ time periods (the right-hand side of the constraint), then the elapsed time of this batch by the end of the

next time interval $[t', t'+1)$ should equal n time periods if it is still executing or is about to complete.

$$\begin{aligned} &\overline{r}_{ijn(t'+1)} = \overline{r}_{ij(n-1)t'}, \\ &\quad \forall j \in \mathbb{J}, i \in \mathbb{I}_j, n \in \{n \in \mathbb{Z}^+ : n \geqslant 2 \wedge n \leqslant \tau_{ijnt'}\}, t' \in \{t, t+1, \cdots, t+h_t\}. \end{aligned} \tag{13}$$

Lifting constraints for batch size. Similarly to Equation (13), the lifting constraints for the batch size are presented below in Equation (14).

$$\begin{aligned} &\overline{b}_{ijn(t'+1)} = \overline{b}_{ij(n-1)t'}, \\ &\quad \forall j \in \mathbb{J}, i \in \mathbb{I}_j, n \in \{n \in \mathbb{Z}^+ : n \geqslant 2 \wedge n \leqslant \tau_{ijnt'}\}, t' \in \{t, t+1, \cdots, t+h_t\}. \end{aligned} \tag{14}$$

Material balance constraints. Equation (15) represents the material balance constraints, which indicate how inventory levels evolve. Here, the term (i) represents the quantity of material k that is produced by upstream batches. To explain, assume that we have $n < \tau_{ijnt'}$, which means that an upstream batch is not completed by the end of $[t'-1, t')$. As a result, we have $1 - \min\{\tau_{ijnt'} - n, 1\} = 0$. Thus, this quantity will not be added to the inventory level for the next time interval. Otherwise we have $n = \tau_{ijnt'}$, which represents the situation that this batch is completed by the end of $[t'-1, t')$, then we have $1 - \min\{\tau_{ijnt'} - n, 1\} = 1$. This quantity will be added to the inventory for the next time interval. In addition to the term (i), the term (ii) represents the quantity of materials consumed by downstream batches. The variable $\overline{p}_{kt}$ represents the quantity of raw materials purchased and the variable $\overline{q}_{kt}$ represents the quantity of products sold. For material $k \notin \mathbb{K}^{\mathrm{R}}$, we manually set $\overline{p}_{kt} = 0$. Also, for material $k \notin \mathbb{K}^{\mathrm{P}}$, we set $\overline{q}_{kt} = 0$.

$$\begin{aligned} \overline{s}_{k(t'+1)} = \overline{s}_{kt'} &+ \underbrace{\sum_{j \in \mathbb{J}} \sum_{i \in \mathbb{I}_j \cap \mathbb{I}_k^+} \sum_{n \in \{n \in \mathbb{Z}^+ : n \geqslant 1 \wedge n \leqslant \tau_{ijnt'}\}} (1 - \min\{\tau_{ijnt'} - n, 1\}) \cdot \rho_{ik} \cdot \overline{b}_{ijnt'}}_{(i)} \\ &+ \underbrace{\sum_{j \in \mathbb{J}} \sum_{i \in \mathbb{I}_j \cap \mathbb{I}_k^-} \rho_{ik} \cdot \overline{y}_{ijt'}}_{(ii)} + \overline{p}_{kt'} - \overline{q}_{kt'}, \\ &\forall k \in \mathbb{K}, t' \in \{t, t+1, \cdots, t+h_t\}. \end{aligned} \tag{15}$$

Constraints for unmet demand. Equation (16) represents the constraints for unmet demand. The unmet demand for the material k during $[t', t'+1)$ equals the unmet demand during the previous time interval minus the quantity sold. It is noteworthy that, although the concept of unmet demand is similar to the concept of backlog (which is presented in Equation (7)), they are actually different in our formulation. While the unmet demand $\overline{m}_{kt'}$ represents the quantity of demand that still remains to be satisfied from t' to $t + h^V$, the backlog l_{kt} represents the accumulative demand that needs to be completed.

$$\overline{m}_{k(t'+1)} = \overline{m}_{kt'} - \overline{q}_{kt'}, \quad \forall k \in \mathbb{K}^{\mathrm{P}}, t' \in \{t, t+1, \cdots, t+h_t\}. \tag{16}$$

Logical constraints between elapsed time and the initiation of a batch. Equation (17) represents the constraints that establish the logic between $\overline{x}_{ijt'}$ and $\overline{r}_{ijnt'}$. When there is a batch initiated at time point t', the elapsed time of this batch should equal one time period by the end of $[t', t'+1)$. Otherwise, when there is no batch initiated at time point t', we should have $\overline{r}_{ij1(t'+1)} = 0$.

$$\overline{r}_{ij1(t'+1)} = \overline{x}_{ijt'}, \quad \forall j \in \mathbb{J}, i \in \mathbb{I}_j, t' \in \{t, \cdots, t+h_t\} \tag{17}$$

Logical constraints between current batch size and input batch size. Similarly to Equation (17), Equation (18) represents the logical constraints between $\overline{y}_{ijt'}$ and $\overline{b}_{ij1(t'+1)}$.

$$\overline{b}_{ij1(t'+1)} = \overline{y}_{ijt'}, \quad \forall j \in \mathbb{J}, i \in \mathbb{I}_j, t' \in \{t, \cdots, t+h_t\}. \tag{18}$$

Machine occupancy constraints. Equation (19) represents the machine occupancy constraints. A machine cannot be occupied by two batches at the same time point t'. Here, the term (i) indicates whether there is a batch initiated at time point t. The term (ii) indicates whether there is a batch that is still executing but not completed by the end of $[t'-1, t')$. By constraining that the sum of (i) and (ii) cannot exceed 1, we ensure that a machine is either idle or occupied by only one batch.

$$\underbrace{\sum_{i \in \mathbb{I}_j} x_{ijt'}}_{(i)} + \underbrace{\sum_{i \in \mathbb{I}_j} \sum_{n \in \{n \in \mathbb{Z}^+ : n \geqslant 1 \wedge n \leqslant \tau_{ijnt'}\}} \overline{r}_{ijnt'}}_{(ii)} \leqslant 1, \quad \forall j \in \mathbb{J}, t' \in \{t, \cdots, t+h_t\} \tag{19}$$

Upper and lower bound constraints. Equations (20)–(22) represent the bounding constraints. Here, Equation (20) represents the lower and upper bounds for raw material purchase. The parameter $p_{kt}^{\max}$ represents the maximum allowable quantity for purchasing material k at time point t. Equation (21) represents the minimum and maximum storage capacity. Equation (22) represents the minimum and maximum batch sizes of initiating a batch.

$$0 \leqslant \overline{p}_{kt'} \leqslant p_{kt}^{\max}, \quad \forall k \in \mathbb{K}^{\mathrm{R}}, t' \in \{t, \cdots, t+h_t\}. \tag{20}$$

$$0 \leqslant \overline{s}_{kt'} \leqslant s_{k}^{\max}, \quad \forall k \in \mathbb{K}, t' \in \{t, \cdots, t+h_t\}. \tag{21}$$

$$y_{ij}^{\min} \leqslant \overline{y}_{ijt'} \leqslant y_{ij}^{\max}, \quad \forall j \in \mathbb{J}, i \in \mathbb{I}_j, t' \in \{t, \cdots, t+h_t\}. \tag{22}$$

Logical constraints for indicators. Equation (23) represents the constraints that establish the logic between $\overline{z}_{t'}$ and $\overline{m}_{kt'}$. If there exist unmet demands during the time interval $[t'-1, t')$ (meaning that $\overline{m}_{kt'} > 0$ for some k), then $\overline{z}_{t'}$ is enforced to equal to 1, which indicates that the demands are still unmet during $[t'-1, t')$. If we arrive at $\overline{m}_{kt'} = 0$, then we have $\overline{z}_{t'} = 0$.

$$M \cdot (\overline{z}_{t'} - 1) \leqslant \sum_{k \in \mathbb{K}^{\mathrm{P}}} \overline{m}_{t'} \leqslant M \cdot \overline{z}_{t'}, \quad \forall t' \in \{t, \cdots, t+h_t\} \tag{23}$$

Logical constraints for makespan. Equation (24) defines a lower bound for the makespan ($\overline{ms}$). If the demand has already been satisfied during the time interval $[t'-1, t')$ (i.e., $\overline{z}_{t'} = 0$), then there are no constraints on the lower bound of $\overline{ms}$. However, if we have $\overline{z}_{t'} = 1$, then $\overline{ms}$ is bounded below by $t' - t$. By checking all possible t' within the scheduling horizon, the minimum value of $\overline{ms}$ is bounded by the actual makespan, which starts from the time point t and ends at the time point t'.

$$\overline{ms} \geqslant M \cdot (\overline{z}_{t'} - 1) + t' - t, \quad \forall t' \in \{t, \cdots, t+h_t\} \tag{24}$$

3.2.3. Objective Functions

In this work, we solve the MILP formulation through a two-stage hierarchical approach. Specifically, in the first stage, we solve the following MILP to obtain the optimal makespan.

$$\begin{aligned} &\min \ \overline{ms} \\ &\text{subject to Equations (9)–(24)} \end{aligned} \tag{25}$$

After solving the first stage, we obtain the optimal makespan, denoted as $\overline{ms}^*$. Then, in the second stage, we solve the following MILP problem.

$$\begin{aligned} &\min \sum_{k\in\mathbb{K}} \sum_{t'=t}^{t+h_t} \bar{s}_{kt'} \\ &\text{subject to} \begin{cases} \text{Equations (9)–(24)} \\ \overline{ms} \leqslant \overline{ms}^* \end{cases} \end{aligned} \tag{26}$$

Intuitively, we solve the second-stage MILP problem to reduce the total inventory level as much as possible. There are two main purposes to solve the scheduling problem in this hierarchical approach. First, in multipurpose batch processes, it is common that there are multiple optimal solutions for an optimal makespan. While some solutions do not cause the inventory to accumulate, some solutions that accumulate the inventory level may result in instability in the long-term. Therefore, it is necessary to minimise the inventory accumulation. Second, in some cases, the magnitude of two different objectives (e.g., makespan and inventory levels) are quite different. If we simply incorporate these two different objectives in a single-stage MILP problem, it may result in numerical issues.

4. Methodology

In this section, we present the detailed description of our rescheduling method. As our method is motivated by a combination of several ideas, and these ideas are reflected in different algorithm modules, it is clearer to present our method in a general-to-specific organisation. In Section 4.1, we first introduce some notations that are used throughout different algorithms. Then, in Section 4.2, we present the overall procedure of the dynamic scheduling process, including how to initialise the system, how to use the dynamic system to describe the evolution of dynamic scheduling, and how to calculate the cumulative nervousness value. In addition, we also briefly introduce the key modules of our rescheduling method, including when- and how-to-reschedule modules, the DAG generating module, and the module that is used to determine the delayable time of an operation. After that, in Sections 4.3–4.5, we expand the technical details of these algorithm modules and justify the motivation behind them.

4.1. Notations

In our method, a schedule is treated as a collection of operations. Here, an operation refers to a batch of a task. In a multipurpose batch scheduling problem, an operation can be sufficiently identified using a 5-tuple, that is, $\mathcal{O} = (i, j, b, t_s, t_f)$. Here, the symbol i represents the task, j represents the machine, b represents the batch size, t_s represents the start time, and t_f represents the completion (finish) time. For example, if there is an initiation of task i_1 on machine j_2 at time point t_5 with 10 units of batch size, and this operation is scheduled to complete at time point t_8, then we write $\mathcal{O} = (i_1, j_2, 10, t_5, t_8)$. When necessary, we use $i(\mathcal{O})$ to denote the task of this operation, $j(\mathcal{O})$ to denote the machine of this operation, $b(\mathcal{O})$ to denote the batch size of this operation, $t_s(\mathcal{O})$ to denote the start time of this operation, and $t_f(\mathcal{O})$ to denote the completion (finish) time of this operation. Additionally, when we have an operation $\mathcal{O}$ as reference, we use it as the index for related variables. For example, consider the variable x_{ijt} that represents the initiation of an operation in the dynamic system (as we presented in Section 3.1). The initiation of operation $\mathcal{O}_1 = (i_1, j_2, 10, t_5, t_8)$ can be simply written as $x_{\mathcal{O}_1}$, which represents $x_{i_1 j_2 t_5}$.

For graph-related notations, we use $\mathcal{G} = (\mathbb{O}, \mathbb{A})$ to denote a DAG, whose nodes represent operations and arcs are connected from one operation to another. When there exists an arc from $\mathcal{O}_1$ to $\mathcal{O}_2$ in the DAG, we denote this structure by $\mathcal{O}_1 \to \mathcal{O}_2$. We use

$\mathrm{Ch}(\mathcal{O})$ to denote the collection of child nodes of an operation $\mathcal{O}$. Namely, for each child node $\mathcal{O}' \in \mathrm{Ch}(\mathcal{O})$, there exists an arc $\mathcal{O}' \to \mathcal{O}$ in the DAG. We also use $\mathrm{De}(\mathcal{O})$ to denote the descendant nodes of an operation $\mathcal{O}$. Namely, for each descendant node $\mathcal{O}' \in \mathrm{De}(\mathcal{O})$, there exists a path $\mathcal{O}' \to \cdots \to \mathcal{O}$ in the DAG.

4.2. Overview

The central idea of our method originates from the observation of how the delay of an operation propagates through a schedule. Generally, a delay in an earlier, upstream operation propagates towards the downstream part of a schedule. However, the specific impact depends on the configuration of production process and the timings of operations. For example, a downstream operation may be completely free from being affected because the delayed upstream operation is not included in any of its prerequisite operations. In other cases, a downstream operation can remain its original start time without being right-shifted or re-assigned to another machine because the idle slots between midstream operations are sufficient to "absorb" the delay disturbance from the upstream operation. Finally, if the downstream operation depends on the delayed upstream operation and the midstream operations are arranged tightly, then the downstream operation may be required to be right-shifted, which extends the original makespan.

Based on this observation, two key questions remain in developing a concrete rescheduling algorithm. These are (1) how to mathematically formalise the propagation of the delay impacts and (2) how to improve the existing methods using this observation. For the first question, we use a Directed Acyclic Graph (DAG) because it is a data structure that represents dependence relationships between a collection of objects. For the second question, by integrating the ideas behind the observation, it turns out that our method improves upon the widely used periodically–completely rescheduling framework mainly in three aspects. First, we reduce the computational time that is required to check whether a rescheduling procedure should be triggered. Second, we use the warm start technique to speed up the solution of MILP problems. Third, we reduce the system nervousness by incorporating the information from the DAG.

The overall procedure of our method is illustrated in Figure 2. At first, we initialise the system with an initial time point, an initial system state, and an empty schedule. Also, we sample the outcomes of processing time variation and intermittent demand that can be observed from the initial time point. After that, we enter the main loop of the dynamic scheduling. While the main loop structure is mostly the same as the periodically–completely rescheduling framework, the key differences lie in the following. (1) Instead of periodically triggering a rescheduling procedure, we propose a novel when-to-reschedule module based on the ideas on the propagation of delay impacts. (2) After a new schedule is generated, rather than naïvely proceeding to the next time point, we also construct a DAG to represent the dependencies between operations in this new schedule. Based on this DAG, we calculate the maximum allowable delay time for each operation and provide the information for the rescheduling trigger in the next several time points. (3) In the how-to-reschedule module, we incorporate information about observed disturbances and the information about the DAG to fix unaffected operations, which can greatly speed up the solution process and eliminate unnecessary operation changes.

For the completeness and ease of implementation, we also provide the pseudo-code of our method in Algorithm 1.

The remainder of this section is dedicated to the explanation of the technical details in each algorithm module. In Section 4.3, we first present how to construct a DAG that represents dependencies between operations in a schedule for a multipurpose batch process. After that, we show how to derive the maximum allowable delay time (that is, the delayable

time of an operation) for each operation in a given DAG. In Section 4.5, we present the when- and how-to-reschedule strategies.

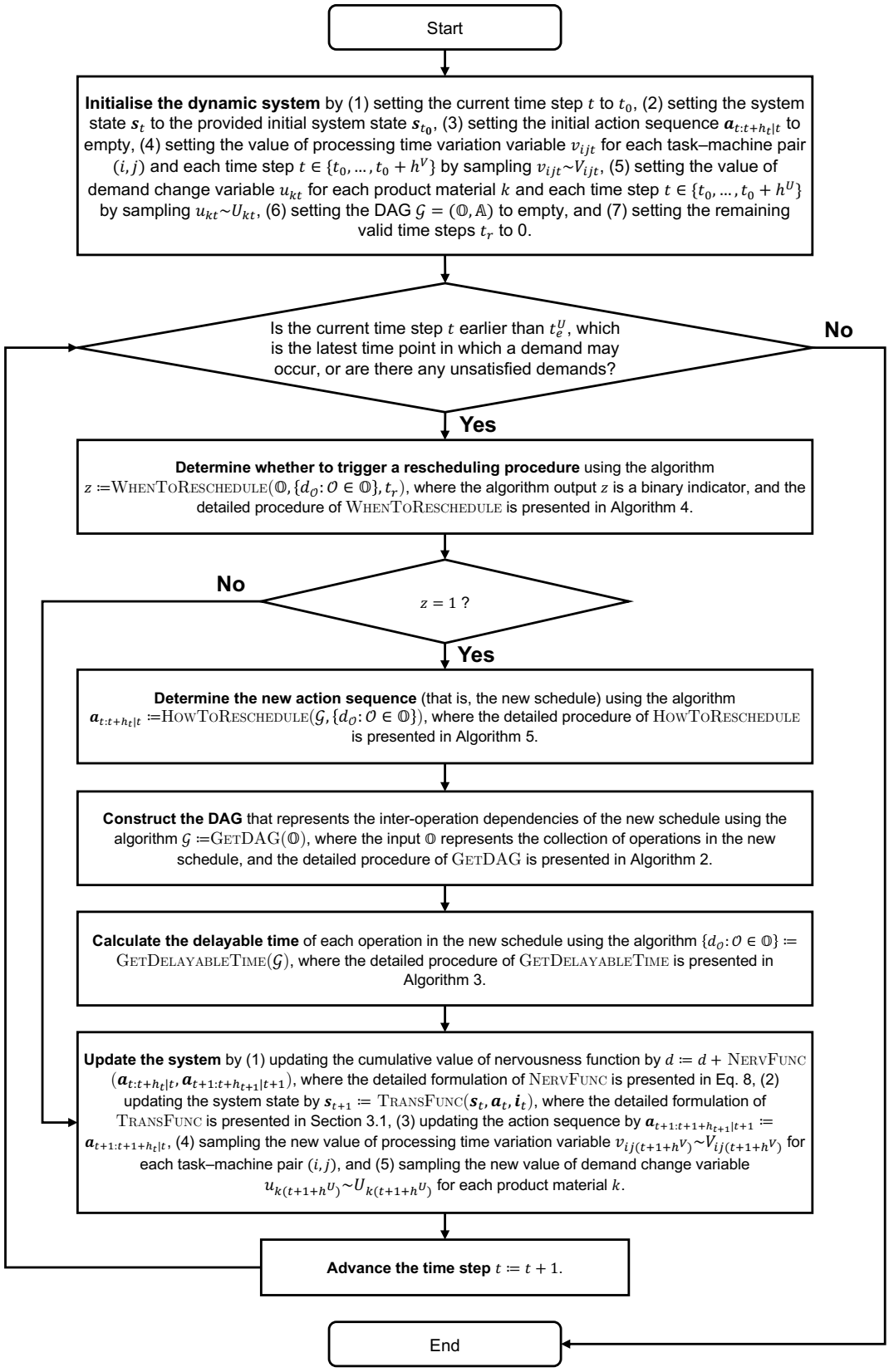


Figure 2. Flowchart of the proposed rescheduling method.

Algorithm 1: Dynamic scheduling of multipurpose batch processes under processing time variation and demand uncertainty

begin
  Initialise $t \coloneqq t_0$, $\boldsymbol{s}_t \coloneqq \boldsymbol{s}_{t_0}$, $\boldsymbol{a}_{t:t+h_t|t} \coloneqq \varnothing$;
  For each $j \in \mathbb{J}$, $i \in \mathbb{I}_j$, $t \in \{t_0, \cdots, t_0 + h^V\}$, initialise $v_{ijt} \sim V_{ijt}$;
  For each $k \in \mathbb{K}^{\mathrm{P}}$ and $t \in \{t_0, \cdots, t_0 + h^U\}$, initialise $u_{kt} \sim U_{kt}$;
  Initialise $\mathbb{O} \coloneqq \varnothing$, $\mathcal{G} \coloneqq (\mathbb{O}, \varnothing)$, and $t_r \coloneqq 0$;
  while $t < t_e^U \vee \sum_{k \in \mathbb{K}^{\mathrm{P}}} l_{kt} > 0$ **do**
    $z \coloneqq \textsc{WhenToReschedule}(\mathbb{O}, \{d_{\mathcal{O}} : \mathcal{O} \in \mathbb{O}\}, t_r)$; `// See Algorithm 5`
    if $z = 1$ **then**
      $\boldsymbol{a}_{t:t+h_t} \coloneqq \textsc{HowToReschedule}(\mathcal{G}, \{d_{\mathcal{O}} : \mathcal{O} \in \mathbb{O}\})$; `// Algorithm 6`
      $\mathcal{G} = (\mathbb{O}, \mathbb{A}) \coloneqq \textsc{GetDAG}(\mathbb{O})$; `// See Algorithm 3`
      $\{d_{\mathcal{O}} : \mathcal{O} \in \mathbb{O}\} \coloneqq \textsc{GetDelayableTime}(\mathcal{G})$; `// See Algorithm 4`
    end
    Update $d \coloneqq d + \textsc{NervFunc}(\boldsymbol{a}_{t:t+h_t|t}, \boldsymbol{a}_{t+1:t+h_{t+1}|t+1})$; `// See Equation (8)`
    Update $\boldsymbol{s}_{t+1} \coloneqq \textsc{TransFunc}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{i}_t)$; `// See Section 3.1`
    Update $\boldsymbol{a}_{t+1:t+1+h_{t+1}|t+1} \coloneqq \boldsymbol{a}_{t+1:t+1+h_t|t}$;
    For each $j \in \mathbb{J}$ and $i \in \mathbb{I}_j$, update $v_{ij(t+1+h^V)} \sim V_{ij(t+1+h^V)}$;
    For each $k \in \mathbb{K}^{\mathrm{P}}$, update $u_{k(t+1+h^U)} \sim U_{k(t+1+h^U)}$;
    Update $t \coloneqq t + 1$;
  end
end
return t, d

4.3. Describing a Schedule Using a Directed Acyclic Graph

At the core of our rescheduling strategy is using a DAG to represent the dependence relationships between different operations. In this DAG, each node represents an operation, and each arc represents a dependency between two operations. As mentioned before, the purpose of constructing such a DAG is to quickly check whether the current makespan will be affected when there are one or more operations are delayed. From this perspective, the DAG that is constructed in this section can be viewed as a model that encodes the knowledge about the execution situation of a schedule.

Essentially, there are two types of dependence relationships between two operations in a schedule for a multipurpose batch process. The first type is *temporal dependence*, which means that, for two consecutive operations processed on the same machine, the latter operation depends on the earlier operation. Here, the term "temporal" comes from the fact that the direction of this type of dependency is along the time axis. Specifically, in our context, by mentioning "dependence", we mean that if the earlier operation is delayed, then the latter operation will also be delayed or at least affected. To describe this type of dependency, we simply add an arc between every pair of two consecutive operations on the same machine in the DAG. This procedure of adding temporal arcs are presented in Lines 7–8 in Algorithm 3.

The second type of dependence relationship is *spatial dependence*, which means that a downstream operation $\mathcal{O}_2$ will depend on an upstream operation $\mathcal{O}_1$, which produces the input material for $\mathcal{O}_2$. We call this type of dependency "spatial" because these dependencies are between different machines. In other words, the direction of spatial dependencies are along the space axis. However, compared to temporal dependencies, spatial dependencies are more subtle to identify. The reason is that in a multipurpose batch process, material

splitting and mixing is allowed in the system. It is possible that the material produced by an upstream operation is used by multiple downstream operations (splitting). Also, the input material of a downstream operation may originate from two or even more upstream operations (mixing). For cases where some materials do not have an intermediate storage tank (i.e., zero storage capacity) and batches have to be transferred to downstream machines immediately after production (i.e., zero-wait policy), the situation will become even more complex.

Despite these difficulties, in this work, we identify spatial dependencies using a backward procedure, which is simple but sufficient for a wide variety of cases. The idea of the backward procedure is as follows. Let $\mathcal{O}$ be an operation. Then, for the material k that is required for initiating $\mathcal{O}$, we track the latest operation(s) in the schedule such that this latest operation(s) produces an enough quantity of k for initiating $\mathcal{O}$. To formalise, we present this backward procedure to determine the spatially upstream operation(s) of an operation in Algorithm 2. An illustrative example of Algorithm 2 can be found in the Supplementary Materials.

By integrating Algorithm 2 and the aforementioned method of adding temporal arcs, we present Algorithm 3. Essentially, Algorithm 3 takes a schedule in the form of a set of operations as the input and outputs a DAG that represents the dependencies between operations in this schedule. The overall procedure of Algorithm 3 is given as follows. For each machine (Line 4), we traverse all the operations that are executed on this machine and add temporal arcs (Lines 7–8) and spatial arcs (Lines 9–10) to the DAG. Finally, we return the DAG with these arcs added (Line 14).

An illustrative example of Algorithm 3 can be found in the Supplementary Materials.

4.4. Calculating the Delayable Time of an Operation

Given a schedule in the form of a set of operations, a DAG constructed using Algorithm 3, and the current makespan of this schedule, we need to know how to determine the delayable time of each operation in this schedule without affecting the current makespan. In other words, we are interested in how many time periods are allowed to postpone an operation without extending the makespan.

We determine the delayable time of an operation using a backward propagation procedure. This procedure is inspired by the concept of the *critical path* [30], which is widely used in the area of job shop scheduling to describe the route of operations that is tight with respect to the makespan. In the context of multipurpose batch processes, it is also possible to find a "generalised critical path". Here, we add the term "generalised" because in multipurpose batch processes, one operation may have multiple spatially upstream or downstream operations due to the existence of batch splitting and mixing. This situation is different from the case in job shop scheduling, where one operation in a job is allowed to have only one spatially upstream or downstream operation because each job has a predetermined operation sequence.

The detailed backward propagation procedure is presented in Algorithm 4. In this procedure, we start from the set of leaf nodes in the DAG (Line 5). For each leaf node, we simply calculate its delayable time as the gap between the current completion time of the schedule and the completion time of this leaf operation (Line 6). Then, for the remaining nodes, we enter a loop (Lines 7–12). In each iteration in this loop, we select an operation and calculate the delayable time of its child nodes (Line 9). We then propagate the delayable time for this operation, as presented in Line 10. This procedure is repeated until all operations in the schedule have been traversed.

Algorithm 2: Backward procedure for determining spatially upstream operations

Function GETSPATUPSTREAMOPER (
$\mathcal{O}$ // Downstream operation
$\mathbb{O}$ // Set of operations in a schedule
)
  begin
    Initialise $\mathbb{O}' := \varnothing$ as the set of spatially upstream operations of $\mathcal{O}$;
    Initialise $\mathbb{K}^-_{\mathcal{O}}$ as the set of input materials of $\mathcal{O}$;
    for each $k \in \mathbb{K}^-_{\mathcal{O}}$ **do**
      Calculate $y_k := |\rho_{i(\mathcal{O}),k} \cdot b(\mathcal{O})|$ as the quantity of material k that is required to initiate $\mathcal{O}$;
      Assign $\mathbb{O}^{\leqslant}_k := \{\mathcal{O}' \in \mathbb{O} : \rho_{i(\mathcal{O}'),k} > 0, t_b(\mathcal{O}') \leqslant t_a(\mathcal{O})\}$. In other words, $\mathbb{O}^{\leqslant}_k$ represents the set of operations that produce material k and complete earlier than the start time of $\mathcal{O}$;
      Assign $y' := 0$ as the cumulative quantity of material that is produced by operations in $\mathbb{O}^{\leqslant}_k$;
      while $y' \leqslant y_k$ **do**
        Assign $\mathcal{O}' := \max_{t_b(\mathcal{O}')} \mathbb{O}^{\leqslant}_k$. In other words, $\mathcal{O}'$ is the operation in $\mathbb{O}^{\leqslant}_k$ with the latest completion time;
        Update $\mathbb{O}' := \mathbb{O}' \cup \{\mathcal{O}'\}$;
        Update $\mathbb{O}^{\leqslant}_k := \mathbb{O}^{\leqslant}_k \setminus \{\mathcal{O}'\}$;
        Update $y' := y' + b(\mathcal{O}')$;
      end
    end
  end
  return $\mathbb{O}'$

Algorithm 3: Constructing a DAG that represents dependencies between operations

Function GETDAG (
$\mathbb{O}$ // Set of operations in a schedule
)
  begin
    Initialise $\mathbb{A} := \varnothing$ as the set of arcs in the desired DAG;
    for each $j \in \mathbb{J}$ **do**
      Assign $\mathbb{O}_j := \{\mathcal{O} \in \mathbb{O} : j(\mathcal{O}) = j\}$. In other words, $\mathbb{O}_j$ represents the set of operations that are executed on machine j;
      for each $\mathcal{O} \in \mathbb{O}_j$ **do**
        Assign $\mathcal{O}' := \max_{t_b(\mathcal{O}')} \mathbb{O}_j$. In other words, $\mathcal{O}'$ represents the immediate temporally upstream operation of $\mathcal{O}$;
        Update $\mathbb{A} := \mathbb{A} \cup \{\mathcal{O} \to \mathcal{O}'\}$;
        Assign $\mathbb{O}' :=$ GETSPATUPSTREAMOPER$(\mathcal{O}, \mathbb{O})$ as the set of spatially upstream operations of $\mathcal{O}$;
        For each $\mathcal{O}' \in \mathbb{O}'$, update $\mathbb{A} := \mathbb{A} \cup \{\mathcal{O} \to \mathcal{O}'\}$;
      end
    end
  end
  return $\mathcal{G} = (\mathbb{O}, \mathbb{A})$

In Algorithm 4, the approach in which we propagate the delayable time (Line 10) is worth explaining. In this formula, we consider a pair of operations, $\mathcal{O}$ and $\mathcal{O}''$. Here, $\mathcal{O}$ represents the parent operation and $\mathcal{O}''$ represents the child operation. By saying "parent" and "child", we mean that in the input DAG $\mathcal{G} = (\mathbb{O}, \mathbb{A})$, there exist an arc $(\mathcal{O}'' \rightarrow \mathcal{O})$ in $\mathbb{A}$. Given the arc $\mathcal{O}'' \rightarrow \mathcal{O}$, we calculate the addition of the delayable time of $\mathcal{O}$ and the time gap between the start time of $\mathcal{O}''$ and the completion time of $\mathcal{O}$. By traversing all the child nodes of $\mathcal{O}$, we then obtain the delayable time of $\mathcal{O}''$. The essence of Algorithm 4 is repeating this process from the leaf nodes and propagates the calculation of delayable time to the entire set of operations in the schedule.

An illustrative example of Algorithm 4 can be found in the Supplementary Materials.

Algorithm 4: Calculating the delayable time

1 **Function** GETDELAYABLETIME (
  $\mathcal{G} = (\mathbb{O}, \mathbb{A})$ `// DAG that represents a schedule`
)
2 **begin**
3 Initialise $t^{\max} := \max_{\mathcal{O} \in \mathbb{O}} t_b(\mathcal{O})$ as the completion time of the schedule. In other words, $t^{\max}$ is the latest completion time of all operations in $\mathbb{O}$;
4 Initialise $\mathbb{O}' := \varnothing$ as the set of operations whose delayable time has already been calculated;
5 Update $\mathbb{O}' := \{\mathcal{O}' \in \mathbb{O} : \mathrm{Ch}(\mathcal{O}') = \varnothing\}$ as the set of leaf nodes in $\mathcal{G}$. In other words, $\mathbb{O}'$ represents the set of operations that have no child nodes;
6 For each operation $\mathcal{O}' \in \mathbb{O}'$, calculate $d_{\mathcal{O}'} := t^{\max} - t_b(\mathcal{O}')$ as the delayable time of $\mathcal{O}'$;
7 **while** $\mathbb{O}' \neq \mathbb{O}$ **do**
8 Assign $\mathbb{O}'' := \{\mathcal{O}' \in \mathbb{O} : \mathrm{Ch}(\mathcal{O}') \in \mathbb{O}', \mathcal{O}' \notin \mathbb{O}'\}$. That is, $\mathbb{O}''$ represents the set of operations whose (1) child nodes are a subset of $\mathbb{O}'$ and (2) delayable time has not been calculated;
9 Select a candidate operation $\mathcal{O}''$ from $\mathbb{O}''$. The specific choice of $\mathcal{O}''$ does not matter;
10 Calculate $d_{\mathcal{O}''} := \min_{\mathcal{O} \in \mathrm{Ch}(\mathcal{O}'')} \left(d_{\mathcal{O}} + (t_a(\mathcal{O}) - t_b(\mathcal{O}''))\right)$. Here, $\mathcal{O}$ represents a child node of $\mathcal{O}''$, $d_{\mathcal{O}}$ represents the delayable time of $\mathcal{O}$, and $t_s(\mathcal{O}) - t_f(\mathcal{O}'')$ represents the gap between the start time of $\mathcal{O}$ and the finish time of $\mathcal{O}''$;
11 Update $\mathbb{O}' := \mathbb{O}' \cup \{\mathcal{O}''\}$;
12 **end**
13 **end**
14 **return** $\{d_{\mathcal{O}} : \mathcal{O} \in \mathbb{O}\}$

4.5. Rescheduling Strategy

4.5.1. When-to-Reschedule Strategy

As mentioned before, one desired property of a rescheduling strategy is computational efficiency. To achieve this, we simply use the logical OR operator to combine a set of simple conditions as the rescheduling trigger. In other words, for a set of logical conditions, if any of these conditions are satisfied, we trigger a rescheduling process. Otherwise, we continue to execute the current schedule.

The set of combined rescheduling conditions is stated as follows. First, based on the observed information related to processing time variation, if we can infer from the set of delayable times that the current makespan will be extended, we then trigger a rescheduling

procedure. In other words, we check whether there exists an operation $\mathcal{O}$ in the current schedule such that $V_{\mathcal{O}} > d_{\mathcal{O}}$. This condition is used to ensure that the system is responsive to critical events that affect the short-term objective.

The second condition is based on the demand response. If we observe new demands, no matter whether they are baseline demands or intermittent demands, we trigger a rescheduling procedure. In other words, for every $t \in \mathbb{T}$, if we have that $\sum_{k \in \mathbb{K}^{\mathrm{P}}} \mu_{k(t+h^U)} + \sum_{k \in \mathbb{K}^{\mathrm{P}}} u_{k(t+h^U)} > 0$, we trigger a rescheduling procedure. It should be noted that, although this heuristic is responsive to any changes in demand, the schedules before and after rescheduling may not appear significantly different. This is because, in our method, triggering a rescheduling procedure is actually equivalent to solving the MILP problem (presented in Section 3.2) using the how-to-reschedule strategy, where many operations are fixed (see the next subsection). As a result, it is often the case where only a few operations are added to or removed from the original schedule when the demand change is slight, rather than triggering a complete rescheduling procedure frequently. However, the cumulative system nervousness may nonetheless be slightly increased in such cases.

The third condition is used to ensure the feasibility of a schedule. Here, we need to introduce a concept, which we call "remaining feasible time steps". We denote the remaining feasible time steps by t_r. Intuitively, t_r represents the forthcoming time steps of a schedule that are guaranteed to be feasible. In our problem, the only source of potential infeasibility comes from the processing time variation. If an upstream operation is delayed, the downstream operations may not be initiated as scheduled. Because whenever we carry out a rescheduling process, we incorporate the information of processing time variation in the form of parameter changes in the MILP problem (see Section 3.2), the first h^V time points are guaranteed to be feasible. As a result, at the time points when a new schedule is generated, we set $t_r := h^V$. When the rescheduling procedure is not triggered, the schedule is advanced by one time step and we reduce t_r by one time period. When we have $t_r = 0$, which means that the remaining schedule is not necessarily guaranteed to be feasible, we trigger a rescheduling procedure.

The formlised when-to-reschedule strategy is presented in Algorithm 5.

Algorithm 5: When-to-reschedule strategy

Function WHENTORESCHEDULE (
$\mathbb{O}$ `// Set of operations in a schedule`
$\{d_{\mathcal{O}} : \mathcal{O} \in \mathbb{O}\}$ `// Set of delayable times`
t_r `// Remaining feasible time steps`
)
  begin
    Assign $z_1 := 1$ if there exist an operation $\mathcal{O} \in \mathbb{O}$ such that $V_{\mathcal{O}} > d_{\mathcal{O}}$. Otherwise, assign $z_1 := 0$;
    Assign $z_2 := 1$ if $\sum_{k \in \mathbb{K}^{\mathrm{P}}} u_{k(t+h^V)} + \sum_{k \in \mathbb{K}^{\mathrm{P}}} \mu_{k(t+h^V)} > 0$. Otherwise, assign $z_2 := 0$;
    Assign $z_3 := 1$ if $t_r = 0$. Otherwise, assign $z_3 := 0$;
  end
  return $\max\{z_1, z_2, z_3\}$

4.5.2. How-to-Reschedule Strategy

At the core of our how-to-reschedule strategy is *warm start*. When a rescheduling procedure is triggered, we solve the MILP presented in Section 3.2 with a specific subset of decision variables fixed. This specific subset of variables includes (1) the initial system state

variables, which are presented in Equations (9)–(12) in Section 3.2, and (2) an additional collection of $\overline{x}_{\mathcal{O}}$ variables (recall that this variable represents the binary decision of initiating an operation) that is associated with the set of operations that are not affected by an event of processing time variation. To obtain the collection (2), let the set $\mathbb{O}$ denote operations that are currently executing or still unexecuted in the original schedule. Then, we identify $\mathbb{O}'$ as the set of operations that are directly affected by an event of processing time variation. In other words, $\mathbb{O}' := \{\mathcal{O}' \in \mathbb{O} : v_{\mathcal{O}'} > 0\}$. After that, for each $\mathcal{O}' \in \mathbb{O}'$, if we determine from the pre-calculated delayable time that $v_{\mathcal{O}'} > d_{\mathcal{O}'}$ (meaning that this operation delay event will affect the current makespan), then we allow operations in $\{\mathcal{O}'\} \cup \mathrm{De}(\mathcal{O}')$ to be revised in the new schedule (recall that $\mathrm{De}(\mathcal{O}')$ represents the set of descendant operations of $\mathcal{O}'$ in the DAG $\mathcal{G} = (\mathbb{O}, \mathbb{A})$). As a formalised version, Algorithm 6 presents the procedure for generating a new schedule with the unaffected operations fixed.

It should be noted that in our how-to-reschedule strategy, for an operation $\mathcal{O}'$ that is affected by an operation delay event, even if this event will not affect the current makespan according to $d_{\mathcal{O}'}$, we still unfix all its descendant operations, namely $\mathrm{De}(\mathcal{O}')$, rather than the operation $\mathcal{O}'$ itself. This is because the procedure used to determine the delayable time of an operation (Algorithm 4) allows all the descendant operations of $\mathcal{O}'$ to be right-shifted. As a result, although the final makespan may not be affected via right-shifting, it is not guaranteed that all operations in $\mathrm{De}(\mathcal{O}')$ will not be right-shifted. In other words, it is possible that some of the descendant operations of $\mathcal{O}'$ are right-shifted without extending the current makespan. Therefore, as a conservative solution, we allow all the descendant operations of $\mathcal{O}'$ to be revised.

Algorithm 6: How-to-reschedule strategy

Function HowToReschedule (
$\mathcal{G} = (\mathbb{O}, \mathbb{A})$ `// DAG that represents the dependencies over the set of executing and un-executed operations in the original schedule`
$\{d_{\mathcal{O}} : \mathcal{O} \in \mathbb{O}\}$ `// Set of delayable times`
)
  begin
    Initialise $\mathbb{O}' := \{\mathcal{O}' \in \mathbb{O} : v_{\mathcal{O}'} > 0\}$. In other words, $\mathbb{O}'$ represents the set of operations that are directly affected by an operation delay event;
    Initialise $\mathbb{O}'' := \varnothing$ as the set of operations that are allowed to be revised in the new schedule;
    for each $\mathcal{O}' \in \mathbb{O}'$ **do**
      if $v_{\mathcal{O}'} > d_{\mathcal{O}'}$ **then**
        Update $\mathbb{O}'' := \{\mathcal{O}'\} \cup \mathrm{De}(\mathcal{O}')$;
      end
      else if $v_{\mathcal{O}'} \leqslant d_{\mathcal{O}'}$ **then**
        Update $\mathbb{O}'' := \mathrm{De}(\mathcal{O}')$;
      end
    end
    Solve the first-stage MILP that was presented in Equation (25) with all variables in $\{\overline{x}_{\mathcal{O}} : \mathcal{O} \in \mathbb{O} \setminus \mathbb{O}''\}$ fixed to value 1;
    Solve the second-stage MILP that was presented in Equation (26) with all variables in $\{\overline{x}_{\mathcal{O}} : \mathcal{O} \in \mathbb{O} \setminus \mathbb{O}''\}$ fixed to value 1;
    Decode the new action set $\boldsymbol{a}'_{t:t+h_t|t}$ from the MIP solution;
  end
  return $\boldsymbol{a}'_{t:t+h_t|t}$

5. Examples

We use two benchmark examples from the literature to evaluate the effectiveness of the proposed rescheduling strategy. The first example (labelled as Example 1) is a medium-sized example adopted from [31]. This example consists of 19 tasks, 8 machines, and 26 types of materials. Figure 3 presents the STN representation of Example 1. Table A1 (see Appendix A) presents the detailed task–machine configurations, which include task–machine compatibilities, the nominal processing time, the minimum and maximum batch sizes, and the probability distribution of the varying processing time. Table A2 (see Appendix A) presents the detailed material configurations, which includes storage capacities, initial inventory levels, supply limits, baseline demands, and the probability distribution of uncertain intermittent demands.

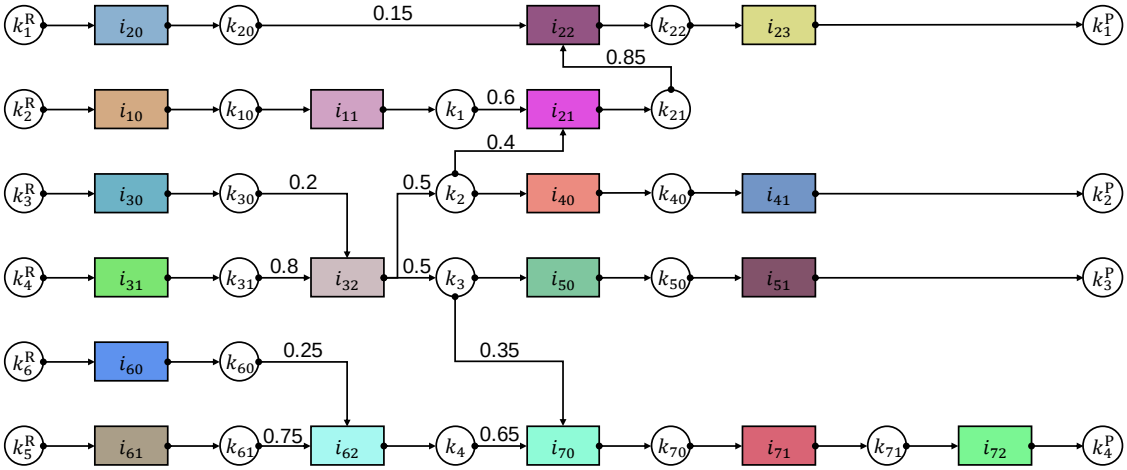


Figure 3. STN representation for the multipurpose batch process in Example 1.

Another example (labelled as Example 2) is adopted from [32], which is a large-sized example with 37 tasks, 12 machines, and 28 types of materials. The STN representation is presented in Figure 4. In their original work, the STN was used to describe a continuous process. However, to be consistent with our theme, we modified this STN to a multipurpose batch process with task–machine and material configurations being randomly generated. These configurations are presented in Tables A3 and A4 in Appendix A.

For the dynamic process, we discretise the entire time axis into uniform time intervals. Each time interval represented one hour time in practice. For both examples, we set $t_e = 1000$ h and $t_e^U = 500$ h. We set $h^V = 12$ h, which means that the value of a varied processing time can be observed 12 h in advance. We also set $h^U = 48$ h, which means that each schedule has to satisfy the forthcoming demand within the next 48 h plus the demands that were backlogged before.

As mentioned before, in the context of dynamic scheduling of multipurpose batch processes, it is generally challenging to have access to or even approximate the optimal rescheduling strategy. Therefore, we compare our method with the periodically–completely rescheduling method in the literature [33–35]. For a fair comparison, we execute the dynamic scheduling process with the same realisation of disturbances. Namely, we use the strategy that solves the MILP formulation presented in Section 3.2 at every time point without fixing any value of decision variables. Although this comparative method is not widely applied in practice, this method generally achieves a very competitive long-term objective. Therefore, we choose to compare our method with this periodically–completely rescheduling strategy.

All our computational results were generated using an Apple MacBook Pro (designed by Apple Inc., Cupertino, CA, USA, assembled in China) with an M1 Pro SoC and 16 GB

of unified memory. The dynamic system module and DAG generation module were implemented in Python 3.9.16. The MILP problems were solved using Gurobi 11.0.2 [36].

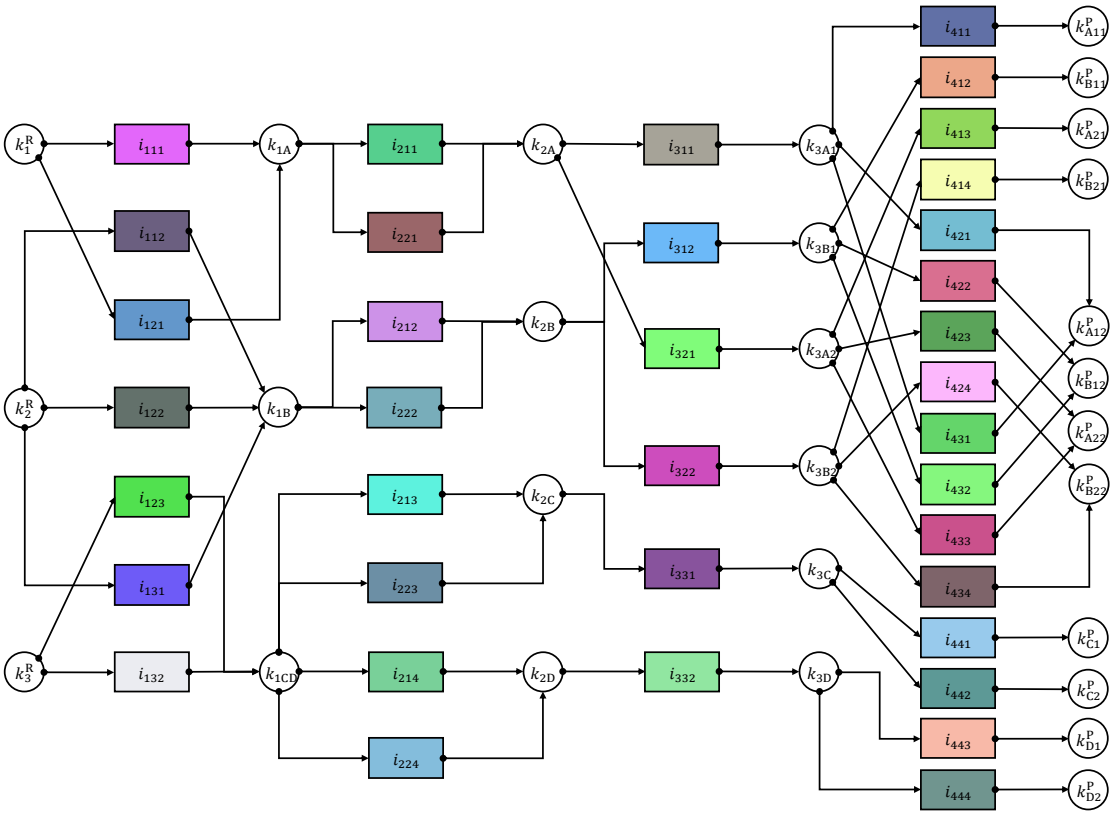


Figure 4. STN representation for the multipurpose batch process in Example 2.

Figure 5 presents the distribution of computational time that is used in solving the MILP problems at each time point. As we can see from Figure 5, the total computational time used by the periodically rescheduling strategy for Example 1 is 4914 s, while our method uses 803 s in total for this example. This gap in computational time is even larger for Example 2. The periodically rescheduling strategy takes 33,554 s to complete the dynamic scheduling process for Example 2, while our method takes 1296 s. This significant difference in computational effort proves the computational efficiency of our method. It is notable that in Example 2, at some time points, the periodic strategy requires a significantly longer computational time to generate a new schedule compared to other time points. For example, we can observe a peak around $t = 200$ and also a peak around $t = 420$. The possible reason for the existence of these peaks is that at some certain time points, a particular realisation of disturbances may have resulted in an MILP problem that is significantly harder to solve. In contrast, there are generally no peaks in the distribution of computational time using our method. This difference in the distribution of computational time proves the effectiveness of the warm start approach.

As mentioned before, the second interest of a rescheduling strategy is the long-term objective. This information is presented in Figure 6. As presented, our method generates a slightly smaller makespan for both examples than the periodically rescheduling strategy. Specifically, the periodically rescheduling strategy completes all demands that arrived within $[t_0, t_{500}]$ for Example 1 using 525 time periods, while our method uses 520 time periods, which was reduced by 0.95%. The overall makespan for Example 2 for the periodically rescheduling strategy is 499 time periods, while it is 490 time periods for ours, which was reduced by 1.80%.

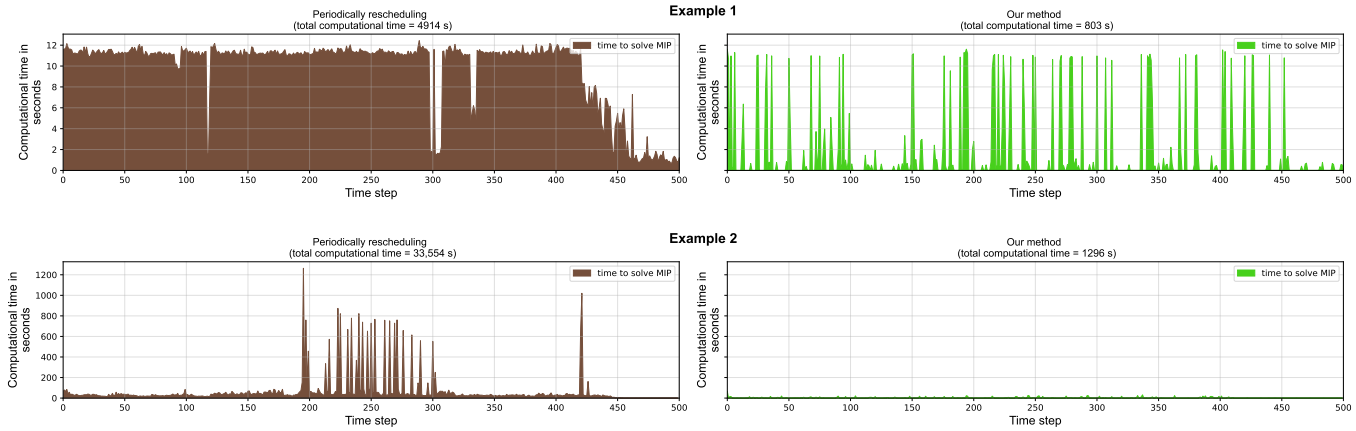


Figure 5. Distribution of computational time that was consumed for solving MILPs at each time step.

As demonstrated by these two examples, our method achieves long-term makespans that are quite close to those achieved by the periodic method. While proving the general effectiveness of our method for the minimisation of long-term makespan requires more numerical experiments, we propose two possible explanations for this closeness. First, from a short-term perspective, our method often generates schedules whose makespan is equal or very close to those generated by the periodic method. This can be seen in Figure 6, where the short-term makespan (green bars) generated by our method is not significantly higher than that from the periodic method (brown bars). Recall that one of the main differences between our method and the periodic method is that our method fixes the start time of certain operations, while the periodic method is not subject to such constraints. Such fixing in our method may have a minimal impact on the short-term makespan. Based on our computational experience, this is true when tasks are not very tightly arranged on a machine in a schedule. This is because there are sufficient idle slots to allow the rearrangement of operations without affecting the makespan. Second, from a long-term perspective, even when our method results in a suboptimal short-term makespan at some time points, the MILP solver can generate schedules where this short-term optimality loss is compensated for in the next few time points. Consequently, these two methods can eventually achieve similar long-term makespans.

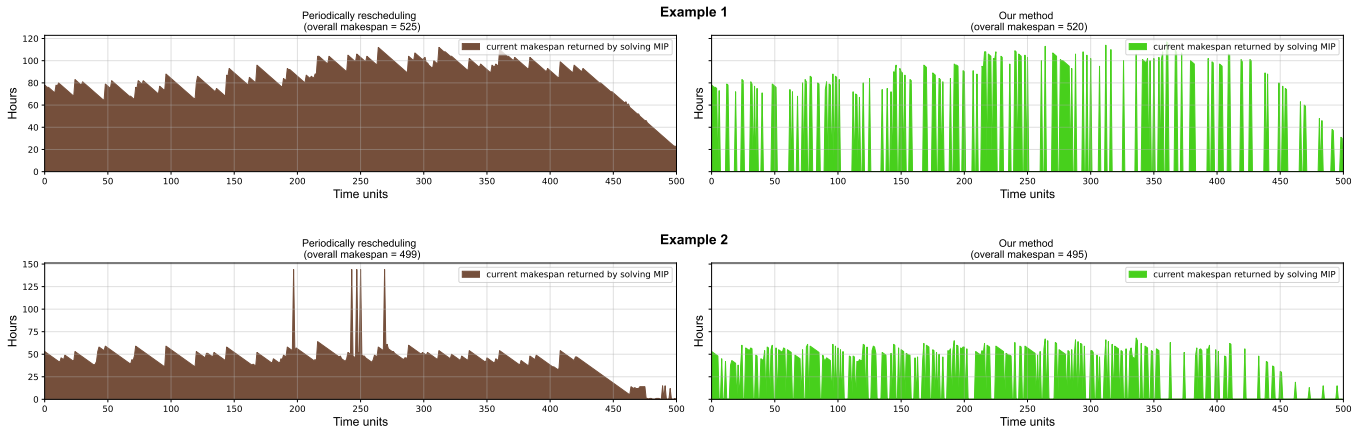


Figure 6. Distribution of current makespan that was returned by solving MILP at each time step.

We are also interested in the cumulative system nervousness. As mentioned in Equation (8), when a rescheduling procedure is triggered, we add the number of operation changes as the value of system nervousness. Figure 7 presents the distribution of the number of operation changes at each time point. In Figure 7, we observe a significant difference in the cumulative system nervousness. The total number of operation changes that is generated by the periodically rescheduling strategy for Example 1 is 41,977 times, while the operation changes from our method during the dynamic scheduling process occurs 5471 times in total, which is reduced by 87.0%. Similarly, the cumulative number of operation changes generated by the periodically rescheduling strategy for Example 2 is 83,474 times, while our method achieves a much lower number, that is, 5627 times, which is reduced by 93.2%. These results show that the cumulative system nervousness can be greatly reduced using the warm start technique.

In brief, the use of our method could achieve a good long-term makespan without changing operations frequently.

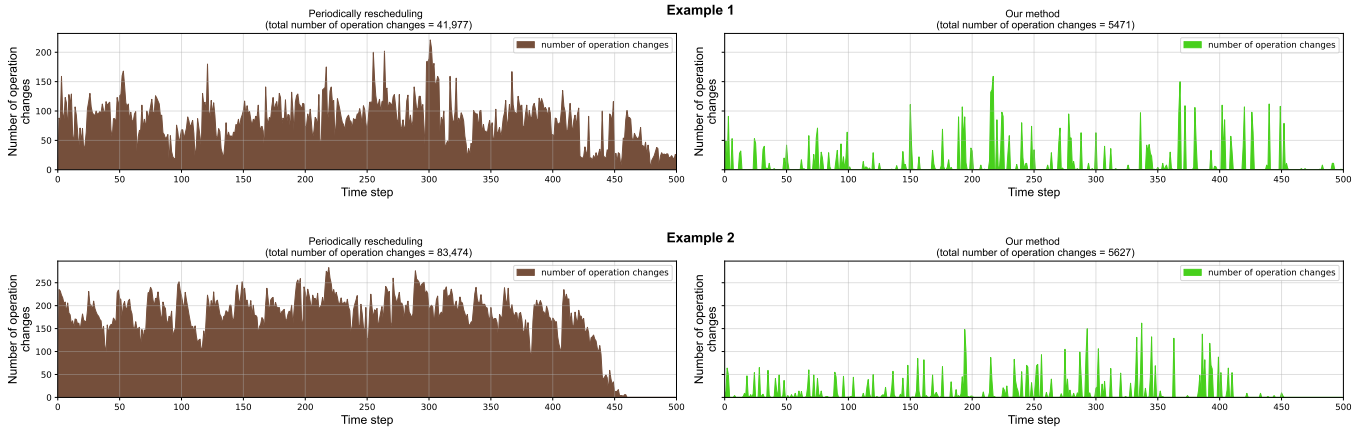


Figure 7. Distribution of number of operation changes at each time step.

6. Conclusions

In this work, we addressed the dynamic scheduling problem of a multipurpose batch process subject to processing time variation and demand uncertainty, which are commonly encountered in industrial practice. While we used a random variable to represent the processing time variation, we assumed that the irregular orders arrived randomly. To model the system dynamics, we used a set of piecewise functions to describe the transition function. We also developed an MILP formulation that was used as a "schedule generator" during the dynamic scheduling process. This MILP formulation was modified from the standard discrete-time formulation for static multipurpose batch process scheduling problems. The difference is that we added a set of initial state constraints to enforce the MILP to be consistent with the current system states. Also, we solved this MILP using a two-stage hierarchical approach to reduce long-term inventory levels.

After introducing the problem formulation, we provided a detailed description of our rescheduling strategy. We first presented how to construct a DAG that describes the inter-operation dependencies that are encoded in a schedule. Specifically, in our method, we identified two types of dependence, namely temporal dependence and spatial dependence. Then, having this DAG established, we presented the procedure for determining how long an operation can be delayed without affecting the current makespan. This method was inspired by the concept of the "critical path" in job shop scheduling. After that, we presented the when- and how-to-reschedule strategies that were used in this paper. Notably, while the when-to-reschedule strategy is simply a set of conditions that are combined by logical ORs, the how-to-reschedule strategy is essentially solving a warm start version of the MILP.

To test our algorithm, we compared our method to the periodically–completely rescheduling strategy with two benchmark examples. The computational results show that our method satisfies the three desired properties that are often expected in dynamic scheduling practice, that is, computational efficiency, goodness in the long-term objective, and a low level of system nervousness.

One of the key limitations of our method is that we considered solely the objective of minimising makespan. While this objective is among the most commonly considered objectives in the scheduling literature, other economic objectives, such as maximising profit and minimising cost, should be considered in future works. Another key limitation is that our method requires a set of rules to determine the dependencies between operations in a schedule. Although in the context of multipurpose batch processes, these rules are not difficult to deduce (as presented in Section 4.3), it may require more elaboration from domain experts to provide these rules when generalising to other production processes. The practicability of providing these rules in other production processes is worth exploring in the future.

In addition, although our method performed well in the presented cases, some remaining issues are worth exploring in the future. The first issue was mentioned in Section 4.3. That is, in this paper, we assumed that the storage policy for materials was finite intermediate storage. However, in other more complex cases, such as zero intermediate storage and zero waiting, the heuristic for determining spatially upstream operations of an operation does not apply. The essence of this difficulty is due to the semi-continuous nature of batch processes. That is, material mixing and splitting are allowed in multipurpose batch processes. Therefore, it is worth exploring how to design the heuristic to determine the spatial dependencies between operations in these cases.

The second issue is related to the warm start. As presented in Algorithm 6, for a specific operation in a schedule, if we observe an event of processing time variation in this operation, then we unfix the start time of all the descendant operations of this affected operation. In our method, we choose this heuristic because it is sometimes not clear which descendant operation requires right-shifting to "absorb" the operation delay event while not affecting the makespan. This heuristic may potentially result in conservativeness. That is, it is possible to select a smaller subset of descendant operations to unfix without affecting the current makespan. However, how to identify this subset is not a trivial task. We will explore this issue in our future work. In addition, we will also address other disturbances such as machine breakdown in our future work.

Supplementary Materials: The following supporting information can be downloaded at https://www.mdpi.com/article/10.3390/pr13020312/s1, Figure S1: STN representation of the illustrative multipurpose batch process; Figure S2: A sample schedule for the illustrative multipurpose batch process; Figure S3: The DAG constructed by Algorithm 3; Figure S4: Visualisation of the result; Table S1: Task-machine configurations; Table S2: Material configurations.

Author Contributions: Conceptualization, T.Z.; Methodology, T.Z., D.L. and J.L.; Software, T.Z.; Formal analysis, T.Z.; Resources, D.L.; Data curation, T.Z. and D.L.; Writing—original draft, T.Z.; Writing—review and editing, T.Z., D.L. and J.L.; Visualization, T.Z.; Supervision, J.L.; Project administration, J.L.; Funding acquisition, J.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by UoM-CSC joint scholarship, (202106440020) and (201908130170) and the Engineering and Physical Sciences Research Council (EP/V051008/1).

Data Availability Statement: The original contributions presented in this study are included in the article/Supplementary Materials. Further inquiries can be directed to the corresponding author.

Acknowledgments: Taicheng Zheng appreciates the financial support from UoM-CSC joint scholarship (202106440020). Dan Li appreciates the financial support from UoM-CSC joint scholarship

(201908130170). Jie Li appreciates the financial support from Engineering and Physical Sciences Research Council (EP/V051008/1).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Throughout this paper, we use the following abbreviations.

APS	Advanced Planning and Scheduling
DAG	Directed Acyclic Graph
ERP	Enterprise Resource Planning
MILP	Mixed-Integer Linear Programming
MES	Manufacturing Execution System
MPC	Model Predictive Control
PSE	Process System Engineering
STN	State-Task Network

Appendix A. System Configurations for Examples

Table A1. Task–machine configurations for Example 1.

i [a]	j [b]	τ_{ij} [c]	$y_{ij}^{\min}$ [d]	$y_{ij}^{\max}$ [e]	p [f]	$[V_{ijt}^{\min}, V_{ijt}^{\max}]$ [g]
i_{20}	j_5	5	2	8	0.1	$[1,3]$
i_{10}	j_1	6	1.5	6	0.1	$[1,3]$
i_{30}	j_4	3	1.75	7	0.1	$[1,3]$
i_{31}	j_3	2	1.75	7	0.1	$[1,3]$
i_{60}	j_4	3	1.75	7	0.1	$[1,3]$
i_{61}	j_6	6	1.5	6	0.1	$[1,3]$
i_{11}	j_7	5	1.75	7	0.1	$[1,3]$
i_{32}	j_2	4	1.25	5	0.1	$[1,3]$
i_{62}	j_8	3	2	8	0.1	$[1,3]$
i_{22}	j_7	3	1.75	7	0.1	$[1,3]$
i_{21}	j_1	4	1.5	6	0.1	$[1,3]$
i_{40}	j_5	5	2	8	0.1	$[1,3]$
i_{50}	j_5	4	2	8	0.1	$[1,3]$
i_{70}	j_6	3	1.5	6	0.1	$[1,3]$
i_{23}	j_4	5	1.75	7	0.1	$[1,3]$
i_{41}	j_2	3	1.25	5	0.1	$[1,3]$
	j_7	2	1.75	7	0.1	$[1,3]$
i_{51}	j_2	2	1.25	5	0.1	$[1,3]$
	j_8	4	2	8	0.1	$[1,3]$
i_{71}	j_8	3	2	8	0.1	$[1,3]$
i_{72}	j_3	2	1.75	7	0.1	$[1,3]$
	j_8	2	2	8	0.1	$[1,3]$

[a] Tasks. [b] Machines. [c] Nominal processing times. [d] Minimum batch sizes. [e] Maximum batch sizes. [f] The probability of an operation being delayed. Specifically, we assume that the sampling process of v_{ijt}, which follows the probability distribution of V_{ijt}, follows a two-step procedure. In the first step, we sample from a Bernoulli distribution with the probability parameter p. The purpose of the first step is to determine whether there is a delay event. For example, if $p = 0.1$, then there is a chance of the probability 0.1 that there is a delay event in the operation of task i on machine j initiated at time point t. [g] The possible duration of a delay event. Continued from table note f, the second step is to determine the duration of the delay event. Here, $V_{ijt}^{\min}$ and $V_{ijt}^{\max}$ represent the minimum and maximum possible duration of the delay, respectively. That is, after we obtain a possible outcome from the Bernoulli distribution in the first stage, we further sample a random value from a discrete uniform distribution that takes integer values within $[V_{ijt}^{\min}, V_{ijt}^{\max}]$.

Table A2. Material configurations for Example 1.

k [a]	$s_k^{\max}$ [b]	s_{kt_0} [c]	$p^{\max}$ [d]	Δp [e]	μ [f]	$\Delta\mu$ [g]	$[u_a, u_b]$ [h]	λ [i]
k_1^{R}	100	0	100	6	-	-	-	-
k_2^{R}	100	0	100	6	-	-	-	-
k_3^{R}	100	0	100	6	-	-	-	-
k_4^{R}	100	0	100	6	-	-	-	-
k_6^{R}	100	0	100	6	-	-	-	-
k_5^{R}	100	0	100	6	-	-	-	-
k_{20}	100	0	-	-	-	-	-	-
k_{10}	20	0	-	-	-	-	-	-
k_{30}	100	0	-	-	-	-	-	-
k_{31}	100	0	-	-	-	-	-	-
k_{60}	20	0	-	-	-	-	-	-
k_{61}	100	0	-	-	-	-	-	-
k_1	100	0	-	-	-	-	-	-
k_2	100	0	-	-	-	-	-	-
k_3	100	0	-	-	-	-	-	-
k_4	100	0	-	-	-	-	-	-
k_{22}	20	0	-	-	-	-	-	-
k_{21}	20	0	-	-	-	-	-	-
k_{40}	20	0	-	-	-	-	-	-
k_{50}	20	0	-	-	-	-	-	-
k_{70}	20	0	-	-	-	-	-	-
k_{71}	100	0	-	-	-	-	-	-
k_{72}	100	0	-	-	1	24	$[1, 3]$	0.01
k_1^{P}	100	0	-	-	3.5	24	$[1, 4]$	0.02
k_2^{P}	100	0	-	-	6.5	24	$[2, 5]$	0.03
k_3^{P}	100	0	-	-	4	24	$[4, 7]$	0.015
k_4^{P}	100	0	-	-	4	24	$[3, 6]$	0.12

[a] Materials. Indices with an "R" superscript represent a raw material. Indices with a "P" superscript represent a product material. The other indices represent intermediate materials. [b] Storage capacities. [c] Initial inventories. [d] Supply upper bound for raw materials. That is, at time points that purchasing raw materials is allowed, the system can at most purchase $p^{\max}$ units of raw materials. [e] Raw material supply intervals. For example, if $\Delta p = 6$, then the system is allowed to purchase raw materials at time points $t_0, t_6, t_{12}, \cdots$. [f] The quantity of each baseline order. [g] The time interval for baseline orders. For example, if $\mu_{k_1^{\mathrm{P}}} = 1$ and $\Delta\mu_{k_1^{\mathrm{P}}} = 24$, then we have regular orders at $t_{24}, t_{48}, \ldots$, with the same quantities of one unit. [h] The lower and upper bounds of the possible quantity of an intermittent order. That is, the quantity of each intermittent order has a continuous uniform distribution with the parameters u_a and u_b. [i] The mean of the Poisson random variable that represents the average number of intermittent orders for the material k. That is, the number of intermittent orders at each time step is represented by a Poisson random variable with the mean λ.

Table A3. Task–machine configurations for Example 2.

i	j	τ_{ij}	$y_{ij}^{\min}$	$y_{ij}^{\max}$	p	$[V_{ijt}^{\min}, V_{ijt}^{\max}]$
i_{111}	j_{11}	3	9	22.5	0.05	[1, 2]
i_{112}	j_{11}	2	6	15	0.05	[1, 2]
i_{121}	j_{12}	3	6	15	0.1	[1, 4]
i_{122}	j_{12}	2	4	10	0.1	[1, 4]
i_{123}	j_{12}	4	8	20	0.1	[1, 4]
i_{131}	j_{13}	2	4	10	0.03	[1, 3]
i_{132}	j_{13}	4	8	20	0.03	[1, 3]

Table A3. *Cont.*

i	j	τ_{ij}	$y_{ij}^{\min}$	$y_{ij}^{\max}$	p	$[V_{ijt}^{\min}, V_{ijt}^{\max}]$
i_{211}	j_{21}	2	6	15	0.08	[2, 3]
i_{212}	j_{21}	2	6	15	0.08	[2, 3]
i_{213}	j_{21}	2	6	15	0.08	[2, 3]
i_{214}	j_{21}	2	6	15	0.08	[2, 3]
i_{221}	j_{22}	2	6	15	0.05	[1, 3]
i_{222}	j_{22}	2	6	15	0.05	[1, 3]
i_{223}	j_{22}	2	6	15	0.05	[1, 3]
i_{224}	j_{22}	2	6	15	0.05	[1, 3]
i_{311}	j_{31}	4	8	20	0.05	[1, 2]
i_{312}	j_{31}	3	6	15	0.05	[1, 2]
i_{321}	j_{32}	4	12	30	0.05	[1, 2]
i_{322}	j_{32}	3	9	22.5	0.05	[1, 2]
i_{331}	j_{33}	2	6	15	0.1	[1, 3]
i_{332}	j_{33}	2	6	15	0.1	[1, 3]
i_{411}	j_{41}	6	12	30	0.04	[2, 4]
i_{412}	j_{41}	6	12	30	0.04	[2, 4]
i_{413}	j_{41}	6	12	30	0.04	[2, 4]
i_{414}	j_{41}	6	12	30	0.04	[2, 4]
i_{421}	j_{42}	5	5	12.5	0.05	[1, 3]
i_{422}	j_{42}	5	5	12.5	0.05	[1, 3]
i_{423}	j_{42}	5	5	12.5	0.05	[1, 3]
i_{424}	j_{42}	5	5	12.5	0.05	[1, 3]
i_{431}	j_{43}	7	10.5	26.25	0.12	[2, 3]
i_{432}	j_{43}	7	10.5	26.25	0.12	[2, 3]
i_{433}	j_{43}	7	10.5	26.25	0.12	[2, 3]
i_{434}	j_{43}	7	10.5	26.25	0.12	[2, 3]
i_{441}	j_{44}	4	8	20	0.04	[1, 3]
i_{442}	j_{44}	4	8	20	0.04	[1, 3]
i_{443}	j_{44}	4	8	20	0.04	[1, 3]
i_{444}	j_{44}	4	8	20	0.04	[1, 3]

Table A4. Material configurations for Example 2.

k	$s_k^{\max}$	s_{kt_0}	$p^{\max}$	Δp	μ	$\Delta\mu$	$[u_a, u_b]$	λ
k_1^{R}	100	35	150	6	-	-	-	-
k_2^{R}	100	45	150	6	-	-	-	-
k_3^{R}	100	35	150	6	-	-	-	-
$k_{1\mathrm{A}}$	24	6	-	-	-	-	-	-
$k_{1\mathrm{B}}$	24	6	-	-	-	-	-	-
$k_{1\mathrm{CD}}$	24	6	-	-	-	-	-	-
$k_{2\mathrm{A}}$	18	4.5	-	-	-	-	-	-
$k_{2\mathrm{B}}$	18	4.5	-	-	-	-	-	-
$k_{2\mathrm{C}}$	18	4.5	-	-	-	-	-	-
$k_{2\mathrm{D}}$	18	4.5	-	-	-	-	-	-
$k_{3\mathrm{A1}}$	32	8	-	-	-	-	-	-
$k_{3\mathrm{B1}}$	32	8	-	-	-	-	-	-
$k_{3\mathrm{A2}}$	32	8	-	-	-	-	-	-
$k_{3\mathrm{B2}}$	32	8	-	-	-	-	-	-
$k_{3\mathrm{C}}$	32	8	-	-	-	-	-	-
$k_{3\mathrm{D}}$	32	8	-	-	-	-	-	-
$k_{\mathrm{A11}}^{\mathrm{P}}$	48	12	-	-	9	24	[4, 10]	0.05
$k_{\mathrm{B11}}^{\mathrm{P}}$	48	12	-	-	13	24	[6, 9]	0.03
$k_{\mathrm{A21}}^{\mathrm{P}}$	48	12	-	-	11	24	[7, 12]	0.05
$k_{\mathrm{B21}}^{\mathrm{P}}$	48	12	-	-	11	24	[6, 8]	0.04
$k_{\mathrm{A12}}^{\mathrm{P}}$	48	12	-	-	10	24	[6, 10]	0.06
$k_{\mathrm{B12}}^{\mathrm{P}}$	48	12	-	-	12	24	[4, 10]	0.03

Table A4. *Cont.*

k	$s_k^{\max}$	s_{kt_0}	$p^{\max}$	Δp	μ	$\Delta\mu$	$[u_a, u_b]$	λ
k^{P}_{A22}	48	12	-	-	9	24	[3, 6]	0.06
k^{P}_{B22}	48	12	-	-	11	24	[6, 8]	0.05
k^{P}_{C1}	48	12	-	-	12	24	[5, 9]	0.06
k^{P}_{C2}	48	12	-	-	8	24	[7, 8]	0.03
k^{P}_{D1}	48	12	-	-	10	24	[5, 6]	0.03
k^{P}_{D2}	48	12	-	-	10	24	[3, 7]	0.05

References

1. Li, D.; Rakovitis, N.; Zheng, T.; Pan, Y.; Li, J.; Kopanos, G. Novel Multiple Time-grid Continuous-time Mathematical Formulation for Short-term Scheduling of Multipurpose Batch Plants. *Ind. Eng. Chem. Res.* **2022**, *61*, 16093–16111. [CrossRef] [PubMed]
2. Susarla, N.; Li, J.; Karimi, I.A. A novel approach to scheduling multipurpose batch plants using unit-slots. *AIChE J.* **2009**, *56*, 1859–1879. [CrossRef]
3. Rakovitis, N.; Zhang, N.; Li, J.; Zhang, L. A new approach for scheduling of multipurpose batch processes with unlimited intermediate storage policy. *Front. Chem. Sci. Eng.* **2019**, *13*, 784–802. [CrossRef]
4. Stefansson, H.; Sigmarsdottir, S.; Jensson, P.; Shah, N. Discrete and continuous time representations and mathematical models for large production scheduling problems: A case study from the pharmaceutical industry. *Eur. J. Oper. Res.* **2011**, *215*, 383–392. [CrossRef]
5. Amaro, A.; Barbosa-Póvoa, A. Planning and scheduling of industrial supply chains with reverse flows: A real pharmaceutical case study. *Comput. Chem. Eng.* **2008**, *32*, 2606–2625. [CrossRef]
6. Kopanos, G.M.; Méndez, C.A.; Puigjaner, L. MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry. *Eur. J. Oper. Res.* **2010**, *207*, 644–655. [CrossRef]
7. Samouilidou, M.E.; Georgiadis, G.P.; Georgiadis, M.C. Food Production Scheduling: A Thorough Comparative Study between Optimization and Rule-Based Approaches. *Processes* **2023**, *11*, 1950. [CrossRef]
8. Palacín, C.G.; Pitarch, J.L.; Vilas, C.; De Prada, C. Integrating Continuous and Batch Processes with Shared Resources in Closed-Loop Scheduling: A Case Study on Tuna Cannery. *Ind. Eng. Chem. Res.* **2023**, *62*, 9278–9289. [CrossRef] [PubMed]
9. Verderame, P.M.; Elia, J.A.; Li, J.; Floudas, C.A. Planning and Scheduling under Uncertainty: A Review Across Multiple Sectors. *Ind. Eng. Chem. Res.* **2010**, *49*, 3993–4017. [CrossRef]
10. Pattison, R.C.; Touretzky, C.R.; Harjunkoski, I.; Baldea, M. Moving horizon closed-loop production scheduling using dynamic process models. *AIChE J.* **2017**, *63*, 639–651. [CrossRef]
11. Gupta, D.; Maravelias, C.T. On the design of online production scheduling algorithms. *Comput. Chem. Eng.* **2019**, *129*, 106517. [CrossRef]
12. Pujawan, I.N. Schedule nervousness in a manufacturing system: A case study. *Prod. Plan. Control* **2004**, *15*, 515–524. [CrossRef]
13. Hwangbo, S.; Liu, J.J.; Ryu, J.H.; Lee, H.J.; Na, J. Production rescheduling via explorative reinforcement learning while considering nervousness. *Comput. Chem. Eng.* **2024**, *186*, 108700. [CrossRef]
14. McAllister, R.D.; Rawlings, J.B.; Maravelias, C.T. Rescheduling Penalties for Economic Model Predictive Control and Closed-Loop Scheduling. *Ind. Eng. Chem. Res.* **2020**, *59*, 2214–2228. [CrossRef]
15. Elkamel, A.; Mohindra, A. A rolling horizon heuristic for reactive scheduling of batch process operations. *Eng. Optim.* **1999**, *31*, 763–792. [CrossRef]
16. Lambrechts, O.; Demeulemeester, E.; Herroelen, W. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *J. Sched.* **2008**, *11*, 121–136. [CrossRef]
17. Subramanian, K.; Maravelias, C.T.; Rawlings, J.B. A state-space model for chemical production scheduling. *Comput. Chem. Eng.* **2012**, *47*, 97–110. [CrossRef]
18. Risbeck, M.J.; Maravelias, C.T.; Rawlings, J.B. Unification of closed-loop scheduling and control: State-space formulations, terminal constraints, and nominal theoretical properties. *Comput. Chem. Eng.* **2019**, *129*, 106496. [CrossRef]
19. Janak, S.L.; Floudas, C.A.; Kallrath, J.; Vormbrock, N. Production Scheduling of a Large-Scale Industrial Batch Plant. II. Reactive Scheduling. *Ind. Eng. Chem. Res.* **2006**, *45*, 8253–8269. [CrossRef]
20. Ferrer-Nadal, S.; Méndez, C.A.; Graells, M.; Puigjaner, L. Optimal Reactive Scheduling of Manufacturing Plants with Flexible Batch Recipes. *Ind. Eng. Chem. Res.* **2007**, *46*, 6273–6283. [CrossRef]
21. Li, Z.; Ierapetritou, M.G. Reactive scheduling using parametric programming. *AIChE J.* **2008**, *54*, 2610–2623. [CrossRef]
22. Vin, J.P.; Ierapetritou, M.G. A New Approach for Efficient Rescheduling of Multiproduct Batch Plants. *Ind. Eng. Chem. Res.* **2000**, *39*, 4228–4238. [CrossRef]

23. Kanakamedala, K.B.; Reklaitis, G.V.; Venkatasubramanian, V. Reactive schedule modification in multipurpose batch chemical plants. *Ind. Eng. Chem. Res.* **1994**, *33*, 77–90. [CrossRef]
24. Honkomp, S.; Mockus, L.; Reklaitis, G. A framework for schedule evaluation with processing uncertainty. *Comput. Chem. Eng.* **1999**, *23*, 595–609. [CrossRef]
25. Gupta, D.; Maravelias, C. A General State-Space Formulation for Online Scheduling. *Processes* **2017**, *5*, 69. [CrossRef]
26. Wittmann-Hohlbein, M.; Pistikopoulos, E.N. Proactive scheduling of batch processes by a combined robust optimization and multiparametric programming approach. *AIChE J.* **2013**, *59*, 4184–4211. [CrossRef]
27. Ikonen, T.J.; Heljanko, K.; Harjunkoski, I. Reinforcement learning of adaptive online rescheduling timing and computing time allocation. *Comput. Chem. Eng.* **2020**, *141*, 106994. [CrossRef]
28. Kondili, E.; Pantelides, C.C.; Sargent, R.W.H. A general algorithm for short-term scheduling of batch operations—I. MILP formulation. *Comput. Chem. Eng.* **1993**, *17*, 211–227. [CrossRef]
29. Shah, N.; Pantelides, C.C.; Sargent, R.W.H. A general algorithm for short-term scheduling of batch operations—II. Computational issues. *Comput. Chem. Eng.* **1993**, *17*, 229–244. [CrossRef]
30. Pinedo, M.L. *Scheduling*; Springer International Publishing: Cham, Switzerland, 2016. [CrossRef]
31. Papageorgiou, L.G.; Pantelides, C.C. Optimal Campaign Planning/Scheduling of Multipurpose Batch/Semicontinuous Plants. 2. A Mathematical Decomposition Approach. *Ind. Eng. Chem. Res.* **1996**, *35*, 510–529. [CrossRef]
32. Maravelias, C.T.; Papalamprou, K. Polyhedral results for discrete-time production planning MIP formulations for continuous processes. *Comput. Chem. Eng.* **2009**, *33*, 1890–1904. [CrossRef]
33. Ikonen, T.J.; Heljanko, K.; Harjunkoski, I. Surrogate-based optimization of a periodic rescheduling algorithm. *AIChE J.* **2022**, *68*, e17656. [CrossRef]
34. Wu, Y.; Maravelias, C.T. A General Model for Periodic Chemical Production Scheduling. *Ind. Eng. Chem. Res.* **2020**, *59*, 2505–2515. [CrossRef]
35. Stevenson, Z.; Fukasawa, R.; Ricardez-Sandoval, L. Evaluating periodic rescheduling policies using a rolling horizon framework in an industrial-scale multipurpose plant. *J. Sched.* **2020**, *23*, 397–410. [CrossRef]
36. Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*; Gurobi Optimization, LLC: Beaverton, OR, USA, 2024.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.