Continuous Production Scheduling MILP Formulations Using Record Keeping Variables

Amin Samadi [a], Christos T. Maravelias [a,b]

[a] Department of Chemical and Biological Engineering, Princeton University, Princeton, NJ 08540

[b] Andlinger Center for Energy and the Environment, Princeton University, Princeton, NJ 08540

ABSTRACT

Most solution methods for mixed-integer linear programming (MILP) production scheduling models have been developed for batch processes. In this paper, we employ integer variables, referred to as record keeping variables (RKVs), into discrete-time continuous production scheduling MILP models that facilitate efficient branching and lead to substantial reductions in solution time. We first introduce different types of RKVs and determine which class of RKVs is the most effective. Second, we explore branching priorities and demonstrate that prioritizing branching on RKVs, relative to other binary variables, leads to further computational improvements. Next, we analyze system attributes, such as task and unit utilization, to determine if prioritizing branching on specific RKVs leads to additional computational enhancements. Our computational results show that the proposed reformulations, in combination with implementing branching priorities, lead to significant computational improvements of continuous production scheduling MILP models.

Keywords

Reformulation, solution method, discrete-time, continuous processes

Introduction

Production scheduling involves generating a schedule of tasks to achieve a specific production objective. Being able to allocate resources in an efficient manner can reduce manufacturing costs, increase facility throughput, and help meet product demands more effectively. Applications for production scheduling range widely from batch production of low-volume products (e.g., pharmaceuticals) to continuous production of high-volume products (e.g., oil and gas refineries)[1–4]. Given the crucial role production scheduling plays in plant operations, it is ubiquitous across industries involving manufacturing. Thus, considerable efforts have been made to develop modeling techniques that account for the diverse process characteristics present in each specific application.

Mathematical optimization is leveraged to assist decision-makers in developing effective production schedules. Production scheduling problems can generally be formulated as mixed-integer linear programming (MILP) models, and due to the specific restrictions of different applications, researchers have developed optimization models that consider various process characteristics including, but not limited to: multiple production environments[5,6], complex storage restrictions[7,8], changeovers[9–13], time-varying pricing[14–16], utility consumption[17], and material transfer restrictions[18]. Accounting for these complex process characteristics enables MILP models to be implemented in a range of industrial applications. However, the development of models that account for multiple characteristics comes with its own challenges: even trivial scheduling instances are computationally expensive. As the number of tasks, units, materials, and time periods increases, instances quickly become intractable[19]. Thus, model complexity is a vital consideration, especially in cases when an instance must be solved numerous times[20,21].

Efforts to reduce the CPU times of production scheduling models have come in the form of decomposition-based algorithms[17,22,23], reformulations[24–28], parallel computing tools[29,30], and tightening methods based on valid inequalities[31–33]. Many of these solution methods, however, are restricted to specific problem types. In this paper, we focus on a simple and easily applicable, reformulation technique that was first introduced by Velez and Maravelias involving the incorporation of integer variables into production scheduling models[24]. Branching on these new integer variables eliminate schedules with the same number of batches, which, in turn, eliminates many symmetric solutions and reduces solution times. More work on these integer variables, termed record keeping variables (RKVs) because they "keep record" of binary variables in the model, was recently performed and additional sets of RKVs were proposed[34]. These RKVs were generated based on: the number of times a task is processed, the number of times a unit processes a task, and the number of tasks performed at a specific time. However, the aforementioned works only focused on batch production scheduling models. The goal of this paper is to assess the strength of a new class of RKVs exclusively defined for continuous production scheduling models, compare them to the original

model, prioritize branching on the RKVs, and rigorously study how branching priorities can impact solution times. More precisely, we assess the impact of prioritizing branching on specific RKVs based on associated system attributes to determine if some RKVs play a more pivotal role in reducing computational resources.

This paper is structured as follows. First, we provide background on production environments and two of the main types of processes. The notation and production scheduling model are reviewed prior to introducing the RKVs. We then present the computational results comparing numerous reformulations as well as solver configurations used to impose branching priorities. By comparing the solution times of the different reformulations and solver configurations, we are able to discuss observations regarding the impact of the RKVs relative to specific instance attributes.

Background

The three main types of production environments, as well as the frameworks that have been used to represent them, are discussed. Then differences between batch and continuous processes are delineated. After the benefits of discrete-time models are detailed, the modeling constraints are introduced.

Continuous Production Scheduling

Production environments are often defined by restrictions on material handling. In sequential production environments, every batch of material follows a specific route, and the mixing or splitting of batches is not allowed since every batch must only be produced/consumed by a single task[35]. In this work, we focus on the more general network environments because they allow for the representation of more complex systems that do not necessarily follow defined stages. In network environments, batch mixing and splitting is permitted, and tasks can produce/consume multiple materials. Material recycling is also permitted. The two most common frameworks utilized to represent network environments are the State-Task Network[9] and Resource-Task Network[36]. Lastly, there are hybrid environments which involve some combination of sequential and network environments. For example, some materials might be produced in a sequential manner where distinct batches are produced but then later get incrementally used as input by multiple batches, as in network environments. A framework able to accurately represent hybrid environments would need to explicitly denote the handling restrictions of every material as well as the processing types of tasks; the General Material Task System was introduced to be able to represent a multitude of material handling restrictions, various process types, transient operations, and even utilities.[37]

Furthermore, most processes can either be classified as batch or continuous. In the former, materials are consumed at the beginning of a process and produced after the process has ended, while

continuous processes produce and consume materials simultaneously during their execution. For batch processes, capacity relates to the size of batches (in kg of material processed), and processing times can be fixed or variable based on the batch size. For continuous processes, capacity implies processing rate (in kg/h of material processed), and processing time corresponds to the duration of the process, which is neither strictly fixed nor dependent upon processing rate. As such, the single degree of freedom of batch processes is batch size, whereas continuous processes have two: processing rate and duration. These distinctions have modeling implications because constraints enforcing fixed processing times are no longer applicable, and additional binary variables are needed to describe the start/end of production, which is no longer fixed. Finally, continuous processes may have significant transient operations, such as startups/shutdowns and direct transitions, which also need to be modeled explicitly.

Model

We employ a discrete-time model (adapted from Wu and Maravelias[38]) because it has been shown to display computational advantages when considering large instances in network environments[39]. For example, they do not incur additional computational costs when material deliveries/shipments occur, and they allow inventory and utility costs to be linearly modeled[39]. Additionally, modeling time-varying utility pricing/availability and time-varying objective function weights requires no new variables or constraints, and production/consumption can occur at intermediate points in time relative to the beginning of a process, not just at the beginning and end[40]. Comprehensive overviews of models employing different time grids, along with in-depth comparisons of their strengths and weaknesses, can be found in the literature[39,41,42].

We define a continuous task as any operation that continuously consumes input materials to produce output materials. All continuous tasks are modeled through consecutive production subtasks that last a single time period; that is, a "subtask" refers to the building blocks of a continuous task, and production subtasks are one type of subtask. Certain continuous tasks require indirect transitions such as startups and shutdowns or direct transitions, which occur between runs of two distinct continuous tasks. These transient operations are also modeled as subtasks. A "run" refers to a string of consecutive production subtasks being executed (i.e., production is continuously occurring), and all subtasks corresponding to the same continuous task produce/consume the same materials (see Figure 1). For simplicity, terms like startup subtasks, shutdown subtasks, and direct transition subtasks will be used synonymously with startups, shutdowns, and direct transitions, respectively.

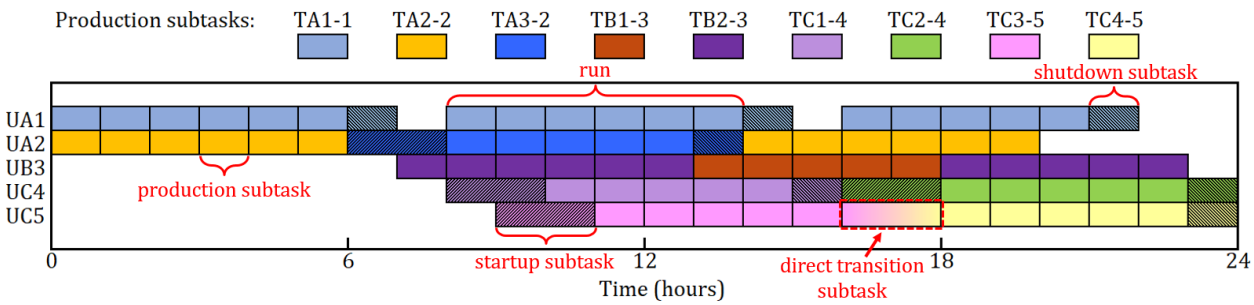


Figure 1. Gantt chart depicting a solution to an instance containing nine production subtasks (shown at the top) and five units (shown on the y-axis) with a time horizon of 24 h. The unit a subtask can be processed in can be found in the subtask name following the hyphen. Production subtasks TC1-4, TC2-4, TC3-5, and TC4-5 have two-hour startups and one-hour shutdowns, while TC3-5 and TC4-5 also have the ability to directly transition to each other.

Fundamental Constraints

Sets and subsets are represented with uppercase bold letters and their respective indices, to denote elements of these sets, with lowercase italic letters. Parameters are represented by lowercase Greek letters, and variables are represented using italic uppercase letters (see Nomenclature). Materials $k \in \mathbf{K}$ are produced/consumed by subtasks $i \in \mathbf{I}$, which must be processed by units $j \in \mathbf{J}$.

The task-unit assignment constraint enforces that a unit can only execute one subtask at a time:

$$\sum_{i \in \mathbf{I}_j} \sum_{t=0}^{\tau_{i,j}-1} X_{i,j,n-t} + \hat{X}^{\mathrm{I}}_{j,n+1} = 1, \qquad \forall\, j \in \mathbf{J}, n \in \mathbf{N}, \tag{1}$$

where binary variable $X_{i,j,n} = 1$ if subtask i is processed by unit j starting at time point n, binary variable $\hat{X}^{\mathrm{I}}_{j,n} = 1$ if unit j is idle during time period n, $\tau_{i,j}$ is the processing time of subtask i processed by unit j, and $\mathbf{I}_j$ is the subset of subtasks i that can be processed by unit j.

The second fundamental constraint is the unit capacity constraint which enforces that processing rates fall between the minimum and maximum allowable processing rates ($\beta^{\mathrm{MIN}}_{i,j}$ and $\beta^{\mathrm{MAX}}_{i,j}$):

$$\beta^{\mathrm{MIN}}_{i,j} X_{i,j,n} \le B_{i,j,n} \le \beta^{\mathrm{MAX}}_{i,j} X_{i,j,n}, \qquad \forall\, i \in \mathbf{I}, j \in \mathbf{J}_i, n \in \mathbf{N}, \tag{2}$$

where the continuous, nonnegative variable $B_{i,j,n}$ is the processing rate of subtask i in unit j starting at time point n. Next, the material balance constraint is:

$$\chi^{\mathrm{MIN}}_k \le S_{k,n+1} = S_{k,n} + \sum_{i \in \mathbf{I}^+_k} \sum_{j \in \mathbf{J}_i} \sum_{t \in \mathbf{T}^+_{i,j}} \rho_{i,k,t} B_{i,j,n-t} + \sum_{i \in \mathbf{I}^-_k} \sum_{j \in \mathbf{J}_i} \sum_{t \in \mathbf{T}^-_{i,j}} \rho_{i,k,t} B_{i,j,n-t} + \xi_{k,n} \le \chi^{\mathrm{MAX}}_k, \tag{3}$$

$$\forall\, k \in \mathbf{K}, n \in \mathbf{N} \backslash \{0\}.$$

where the continuous, nonnegative variable $S_{k,n}$ is the inventory level of material k during time period n; $\mathbf{I}^+_k$ and $\mathbf{I}^-_k$ include the subtasks that produce and consume material k, respectively; $\mathbf{J}_i$ is the

subset of units j that can process subtask i; and $\mathbf{T}_{i,j}^{+} = \{1, 2, \dots, \tau_{i,j}\}$ and $\mathbf{T}_{i,j}^{-} = \{0, 1, \dots, \tau_{i,j} - 1\}$ include time points with respect to the start of subtask i. The parameter $\rho_{i,k,t}$ is the conversion coefficient of material k produced (>0) or consumed (<0) by subtask i after t periods after the start of a subtask. The first summation term in Eq. (3) is positive, and the second summation term is negative according to the sign of $\rho_{i,k,t}$. The $\xi_{k,n}$ parameter is the net amount of material k shipped at time point n ($\xi_{k,n} > 0$ for deliveries arriving to the plant and $\xi_{k,n} < 0$ for orders departing the plant); $\chi_k^{\mathrm{MIN}}/\chi_k^{\mathrm{MAX}}$ are the minimum/maximum inventory capacity for material k.

Transient Operations

Transient operations such as startups, shutdowns, and direct transitions are modeled by:

$$X_{i,j,n-1} - X_{i,j,n} + \sum_{i' \in \mathbf{I}_i^{\mathrm{SU}}} X_{i',j,n-\tau_{i',j}} - \sum_{i' \in \mathbf{I}_i^{\mathrm{SD}}} X_{i',j,n} + \sum_{i' \in \mathbf{I}_i^{\mathrm{DT+}}} X_{i',j,n-\tau_{i',j}} - \sum_{i' \in \mathbf{I}_i^{\mathrm{DT-}}} X_{i',j,n} = 0, \quad \forall\, i \in \mathbf{I}^{\mathrm{SS}} \cup \mathbf{I}^{\mathrm{DT}}, j \in \mathbf{J}_i, n \in \mathbf{N}, \tag{4}$$

where $\mathbf{I}_i^{\mathrm{SU}}/\mathbf{I}_i^{\mathrm{SD}}$ include startups/shutdowns associated with production subtask i, and $\mathbf{I}_i^{\mathrm{DT+}}/\mathbf{I}_i^{\mathrm{DT-}}$ include direct transitions to/from production subtask i. All production subtasks that involve startups or shutdowns are included in subset $\mathbf{I}^{\mathrm{SS}}$, and all production subtasks that have direct transitions associated with them are included in subset $\mathbf{I}^{\mathrm{DT}}$. Eq. (4) enforces that a unit j must process subtask i if the subtask was processed in the previous time period, a startup just took place, or a direct transition to the subtask in question just took place. Conversely, a unit j stops processing production subtask i if a shutdown or direct transition away from the subtask in question just took place.

To properly link transitions to their corresponding production subtask, we use:

$$X_{i,j,n} \geq \sum_{i' \in \mathbf{I}_i^{\mathrm{TR+}}} X_{i',j,n-\tau_{i',j}}, \qquad \forall\, i \in \mathbf{I}^{\mathrm{SS}} \cup \mathbf{I}^{\mathrm{DT}}, j \in \mathbf{J}_i, n \in \mathbf{N}. \tag{5}$$

which enforces that after a transition is executed, the associated production subtask must start being processed.

A constraint to track the activity of a unit is also written:

$$\hat{X}_{j,n}^{\mathrm{I}} + \sum_{i' \in \mathbf{I}_i^{\mathrm{SD}}} X_{i',j,n-\tau_{i',j}} - \sum_{i' \in \mathbf{I}_i^{\mathrm{SU}}} X_{i',j,n} = \hat{X}_{j,n+1}^{\mathrm{I}}, \qquad \forall\, j \in \mathbf{J}^{\mathrm{SS}}, n \in \mathbf{N}, \tag{6}$$

where subset $\mathbf{J}^{\mathrm{SS}}$ includes units involving startups or shutdowns. According to Eq. (6), a unit will start or stop being idle when a shutdown finishes or a startup begins, respectively.

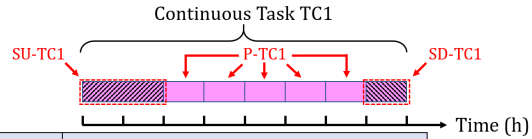
Run-Starting and Run-Ending

A continuous task is modeled using subtasks, and a run is defined as a string of consecutive, single-period production subtasks. To describe the relationship between the start of a run, the end of a run, and subtask execution during the run itself, Eqs. (7) and (8) are needed.

$$Y^{\mathrm{S}}_{i,j,n} = X_{i,j,n} - X_{i,j,n-1} + Y^{\mathrm{E}}_{i,j,n}, \qquad \forall\, i \in \mathbf{I}^{\mathrm{P}}, j \in \mathbf{J}_i, n \in \mathbf{N} \tag{7}$$

$$Y^{\mathrm{S}}_{i,j,n} + Y^{\mathrm{E}}_{i,j,n} \leq 1, \qquad \forall\, i \in \mathbf{I}^{\mathrm{P}}, j \in \mathbf{J}_i, n \in \mathbf{N} \tag{8}$$

Subset $\mathbf{I}^{\mathrm{P}}$ includes all production subtasks, and $Y^{\mathrm{S}}_{i,j,n}$ and $Y^{\mathrm{E}}_{i,j,n}$ equal 1 when a run starts and ends, respectively; $Y^{\mathrm{S}}_{i,j,n}$ and $Y^{\mathrm{E}}_{i,j,n}$ are directly linked to production subtasks and not transitions (see Figure 2). Eq. (8) enforces that both $Y^{\mathrm{S}}_{i,j,n}$ and $Y^{\mathrm{E}}_{i,j,n}$ cannot be equal to 1 during a run (i.e., when $X_{i,j,n}$ and $X_{i,j,n-1}$ both equal one) or during an idle period (i.e., when $X_{i,j,n}$ and $X_{i,j,n-1}$ both equal zero) since this would not accurately portray what is physically occurring in a facility; a run cannot start and end at the same time.



Time Point (n)	1	2	3	4	5	6	7	8	9
$Y^{\mathrm{S}}_{\mathrm{P-TC1},j,n}$	0	0	1	0	0	0	0	0	0
$Y^{\mathrm{E}}_{\mathrm{P-TC1},j,n}$	0	0	0	0	0	0	0	1	0
$X_{\mathrm{P-TC1},j,n}$	0	0	1	1	1	1	1	0	0
$X_{\mathrm{SU-TC1},j,n}$	1	0	0	0	0	0	0	0	0
$X_{\mathrm{SD-TC1},j,n}$	0	0	0	0	0	0	0	1	0

Figure 2. Relevant binary variables are depicted to demonstrate which time points during task execution they correspond to. Continuous task TA3 possesses a 2h starup and 1h shutdown. Note: $Y^{\mathrm{S}}_{i,j,n}$ and$Y^{\mathrm{E}}_{i,j,n}$ do not equal 1 when a startup and shutdown occurs but when the production subtasks start and end, respectively.

Run Length

Minimum and maximum run length constraints can be enforced with Eqs. (9) and (10), respectively.

$$X_{i,j,n} \geq \sum_{n' \in \mathbf{N}^{\mathrm{MIN}}_{i,j,n}} Y^{\mathrm{S}}_{i,j,n'}, \qquad \forall\, i \in \mathbf{I}^{\mathrm{P}}, j \in \mathbf{J}_i, n \in \mathbf{N} \tag{9}$$

$$\tau^{\mathrm{MAX}}_{i,j} \geq \sum_{n' \in \mathbf{N}^{\mathrm{MAX}}_{i,j,n}} X_{i,j,n'}, \qquad \forall\, i \in \mathbf{I}^{\mathrm{P}}, j \in \mathbf{J}_i, n \in \mathbf{N}, \tag{10}$$

where $\tau^{\mathrm{MIN}}_{i,j}/\tau^{\mathrm{MAX}}_{i,j}$ are the minimum/maximum run lengths, $\mathbf{N}^{\mathrm{MIN}}_{i,j,n} = \{n - \tau^{\mathrm{MIN}}_{i,j} + 1, n - \tau^{\mathrm{MIN}}_{i,j} + 2, \dots, n\}$, and $\mathbf{N}^{\mathrm{MAX}}_{i,j,n} = \{n - \tau^{\mathrm{MAX}}_{i,j}, n - \tau^{\mathrm{MAX}}_{i,j} + 1, \dots, n\}$ with $\left|\mathbf{N}^{\mathrm{MAX}}_{i,j,n}\right| = \tau^{\mathrm{MAX}}_{i,j} + 1$.

Objective Function

The objective function to minimize total cost consists of three terms:

$$\min \sum_{k\in\mathbf{K}}\sum_{n\in\mathbf{N}\setminus\{0\}} \gamma^{\mathrm{S}}_{k} S_{k,n} + \sum_{i\in\mathbf{I}}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} (\gamma^{\mathrm{X}}_{i,j,n} X_{i,j,n} + \gamma^{\mathrm{B}}_{i,j,n} B_{i,j,n}) + \sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} (\gamma^{\mathrm{Y}}_{i,j,n} Y^{\mathrm{S}}_{i,j,n}) \qquad (11)$$

The inventory, fixed subtask execution, variable subtask execution, and run-starting costs are $\gamma^{\mathrm{S}}_{k,n}$, $\gamma^{\mathrm{X}}_{i,j,n}$, $\gamma^{\mathrm{B}}_{i,j,n}$, and $\gamma^{\mathrm{Y}}_{i,j,n}$, respectively. The first term calculates the total inventory costs, the second term calculates the fixed and variable costs associated with executing all subtasks, and the third term calculates the costs associated with starting a run.

We make the following assumptions in our model: instance data is deterministic, every subtask ends within the set scheduling horizon (this includes transient operations such as shutdowns), and material transfer between units is instantaneous.

Methods

Although implementing RKVs may seem to trivially increase the number of variables in the model, they appear to also allow solvers to perform branching more efficiently and close the optimality gap faster. In the following sections, we discuss two classes of RKVs that can be generated based on the two types of binary variables present in the model: the variables with the superscript "X" involve the $X_{i,j,n}$ binary variable and those with the superscript "Y" involve the $Y^{\mathrm{S}}_{i,j,n}$ binary variable. We then provide an example of how solvers can take advantage of RKVs to reduce solution times. This technique has been shown to successfully reduce computational resources in batch scheduling problems[34,43], so we aim to study RKVs in the context of continuous production scheduling models and assess their impact on solution times. Moreover, we analyze variable branching and how prioritizing branching on specific RKVs can result in closing the optimality gap even more quickly.

RKVs involving $X_{i,j,n}$

We first introduce $N^{\mathrm{X}}_{i,j}$, which is the sum of the $X_{i,j,n}$ binary variable over all time points.

$$\sum_{n\in\mathbf{N}} X_{i,j,n} = N^{\mathrm{X}}_{i,j} \leq \left\lfloor \frac{|\mathbf{N}|-1}{\tau_{i,j}} \right\rfloor, \qquad \forall\, i\in\mathbf{I}^{\mathrm{P}}, j\in\mathbf{J}_i, \qquad (12)$$

where $N^{\mathrm{X}}_{i,j}$ is bounded by the number of production subtask executions that can feasibly occur over the time horizon; rounding down $\frac{|\mathbf{N}|-1}{\tau_{i,j}}$ yields the maximum number of subtask executions possible. Importantly, the upper bound is introduced because some MILP solvers recognize that RKVs are a summation of other binary variables and substitute them out of the model prior to solving the instance[34]. The introduction of bounds ensures that the RKVs are not substituted. In order to

decouple the effects of the bounds from the RKVs themselves, we tested the model that includes the bounds only (see Supporting Information). The results demonstrate that simply introducing the bounds without any RKVs appears to increase solution times.

We generate other RKVs that keep record of $X_{i,j,n}$ in various ways in order to produce new integer variables that the solver can branch on. Variable N_i^{X} is equal to the summation of $X_{i,j,n}$ over all units and time points, tracking the total number of subtask executions for each production subtask.

$$\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} X_{i,j,n} = N_i^{\mathrm{X}} \le \sum_{j\in\mathbf{J}_i}\left\lfloor\frac{|\mathbf{N}|-1}{\tau_{i,j}}\right\rfloor, \qquad \forall\, i\in\mathbf{I}^{\mathrm{P}} \tag{13}$$

The upper bound is calculated similarly, but a summation is taken over all units because N_i^{X} is written for every production subtask.

Eq. (14) sums $X_{i,j,n}$ over all subtasks and time points, essentially tracking the number of times any production subtask is executed on a particular unit over the time horizon:

$$\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{n\in\mathbf{N}} X_{i,j,n} = N_j^{\mathrm{X}} \le \left\lfloor\frac{|\mathbf{N}|-1}{\min_{i\in\mathbf{I}_j}\{\tau_{i,j}\}}\right\rfloor, \qquad \forall\, j\in\mathbf{J}_i \tag{14}$$

The next RKV we introduce, N_n^{X}, is written for time points, so the number of variables that the model is increasing by correlates to the discretization of the time horizon.

$$\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{j\in\mathbf{J}_i} X_{i,j,n} = N_n^{\mathrm{X}} \le |\mathbf{J}|, \qquad \forall\, n\in\mathbf{N}\backslash\{|\mathbf{N}|\}. \tag{15}$$

The upper bound, $|\mathbf{J}|$, holds true because there cannot be more subtasks actively being processed than the number of available units.

Lastly, we sum $X_{i,j,n}$ over all subtasks, units, and time points to define N^{X}:

$$\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} X_{i,j,n} = N^{\mathrm{X}} \le \min\left\{\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{j\in\mathbf{J}_i}\left\lfloor\frac{|\mathbf{N}|-1}{\tau_{i,j}}\right\rfloor, \sum_{j\in\mathbf{J}_i}\left\lfloor\frac{|\mathbf{N}|-1}{\min_{i\in\mathbf{I}_j}\{\tau_{i,j}\}}\right\rfloor\right\} \tag{16}$$

The reader can deduce how the upper bound in Eq. (16) is calculated by noting the similarities of the bounds in Eqs. (13) and (14). Eq. (16) additionally includes a summation over all production subtasks for the first term within the min{·} function and includes a summation over all units for the second term within the min{·} function. Subsequently, the minimum between the two calculated values serves as the upper bound because the total number of subtask executions over all units and time points cannot be larger than the bounds previously calculated in Eqs. (13) and (14).

RKVs involving $Y^{\mathrm{S}}_{i,j,n}$

We begin introducing the second class of RKVs by starting with $N^{\mathrm{Y}}_{i,j}$, which is the sum of the $Y^{S}_{i,j,n}$ binary variable over all time points.

$$\sum_{n\in\mathbf{N}} Y^{\mathrm{S}}_{i,j,n} = N^{\mathrm{Y}}_{i,j} \le \left\lfloor \frac{|\mathbf{N}|-1}{\tau^{\mathrm{MIN}}_{i,j}+\alpha^{\mathrm{MIN}}_{i,j}} \right\rfloor, \qquad \forall\, i\in\mathbf{I}^{\mathrm{P}}, j\in\mathbf{J}_i, \tag{17}$$

where $\alpha^{\mathrm{MIN}}_{i,j}$ is the minimum time needed to transition to and from a run. Eq. (17) enforces an upper bound based on the maximum number of runs that can be processed over the time horizon. If a production subtask has no transition subtasks associated with it, $\alpha^{\mathrm{MIN}}_{i,j}=0$. Together, $\tau^{\mathrm{MIN}}_{i,j}$ and $\alpha^{\mathrm{MIN}}_{i,j}$ sum up to the minimum amount of time a continuous task requires from startup to shut down.

If the summation of $N^{\mathrm{Y}}_{i,j}$ is taken over all units, we define N^{Y}_{i}:

$$\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} Y^{\mathrm{S}}_{i,j,n} = N^{\mathrm{Y}}_{i} \le \sum_{j\in\mathbf{J}_i} \left\lfloor \frac{|\mathbf{N}|-1}{\tau^{\mathrm{MIN}}_{i,j}+\alpha^{\mathrm{MIN}}_{i,j}} \right\rfloor, \qquad \forall\, i\in\mathbf{I}^{\mathrm{P}} \tag{18}$$

Once again, the upper bound is valid because the maximum number of runs possible for any production subtask is the time horizon divided by $\tau^{\mathrm{MIN}}_{i,j}+\alpha^{\mathrm{MIN}}_{i,j}$ added up for all units $j\in\mathbf{J}_i$.

The next RKV, N^{Y}_{j}, is equal to the summation of $Y^{\mathrm{S}}_{i,j,n}$ over all production subtasks and time points:

$$\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{n\in\mathbf{N}} Y^{\mathrm{S}}_{i,j,n} = N^{\mathrm{Y}}_{j} \le \left\lfloor \frac{|\mathbf{N}|-1}{\min\limits_{i\in\mathbf{I}^{\mathrm{P}}}\{\tau^{\mathrm{MIN}}_{i,j}+\alpha^{\mathrm{MIN}}_{i,j}\}} \right\rfloor, \qquad \forall\, j\in\mathbf{J}_i. \tag{19}$$

The upper bound of N^{Y}_{j} is the largest number of runs that can be processed in a unit j over the time horizon.

We also introduce N^{Y}_{n}, which is the summation of $Y^{\mathrm{S}}_{i,j,n}$ binary variables over all production subtasks and units able to process them.

$$\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{j\in\mathbf{J}_i} Y^{\mathrm{S}}_{i,j,n} = N^{\mathrm{Y}}_{n} \le |\mathbf{J}|, \qquad \forall\, n\in\mathbf{N}\backslash\{|\mathbf{N}|\} \tag{20}$$

For the same reason discussed for Eq. (15), the upper bound of N^{Y}_{n} is the total number of units in the system.

Finally, we take the summation of $Y^{\mathrm{S}}_{i,j,n}$ over all subtasks, units, and time points to yield N^{Y}. This value denotes the total number of runs that occur in all units during the time horizon.

$$\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} Y^{\mathrm{S}}_{i,j,n} = N^{\mathrm{Y}} \leq \min\left\{\sum_{i\in\mathbf{I}^{\mathrm{P}}}\sum_{j\in\mathbf{J}_i}\left\lfloor\frac{|\mathbf{N}|-1}{\tau^{\mathrm{MIN}}_{i,j}+\alpha^{\mathrm{MIN}}_{i,j}}\right\rfloor, \sum_{j\in\mathbf{J}_i}\left\lfloor\frac{|\mathbf{N}|-1}{\min\limits_{i\in\mathbf{I}^{\mathrm{P}}}\{\tau^{\mathrm{MIN}}_{i,j}+\alpha^{\mathrm{MIN}}_{i,j}\}}\right\rfloor\right\} \tag{21}$$

The minimum of the bounds seen in Eq. (18) and (19) are used, but Eq. (21) additionally includes the summation over all production subtasks for the first term within the min{·} function and the summation over all units for the second term within the min{·} function.

Motivating Example

To demonstrate how RKVs allow solvers to branch more efficiently, we use the example in Figure 3 with an integer variable X on the x-axis and an integer variable Y on the y-axis. The feasible region of the linear programming (LP) relaxation is shown as the blue shaded region, and integer feasible points are shown as yellow points. The objective function is to maximize $2X + Y$, and the level curve of the objective function is shown as a purple line. Before any variable branching is performed, the first solution of the LP relaxation is the purple point. A solver would then proceed to branch on an integer variable such as X or Y to reduce the feasible region of the LP relaxation; one such branch could be $X \leq 5 \vee X \geq 6$ (illustrated by the green dotted line). Without eliminating any integer feasible solutions, this branch would allow the solver to yield the green point (5, 1.4), which is closer to the optimal solution (circled in red) at point (5, 1). Note that there is no way to yield the optimal solution after branching on X or Y once; additional branching must occur to reduce the feasible region before being able to yield the optimal solution. However, with the introduction of the RKV denoted as N, which is equal to $X + Y$, the solver has the ability to branch on the new integer variable. Branching on N, specifically $N \leq 6 \vee N \geq 7$, results in eliminating the blue shaded region above the red dotted line. Upon solving the instance again, the solver yields the optimal solution after only branching once.

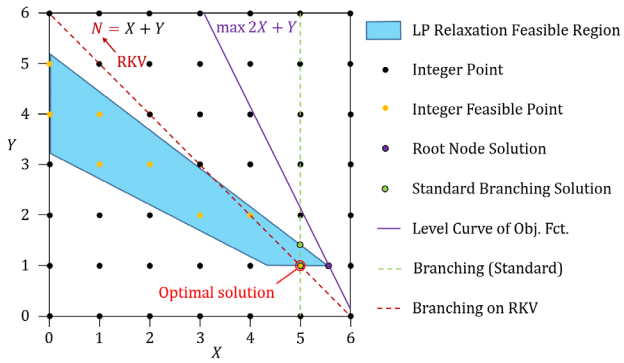


Figure 3. Motivating example illustrating efficient branching on RKVs. The level curve of the objective function (maximize $2X + Y$) is shown to intercept the blue shaded region (feasible region of LP relaxation) at the purple point (root node solution). After branching on the X integer variable to enforce $X \leq 5 \vee X \geq 6$ (shown by a green dotted line), the next solution obtained is at the green point (5, 1.4). However, the presence of the RKV $N = X + Y$ allows the solver to arrive at the optimal solution (5, 1) after only a single branch: $N \leq 6 \vee N \geq 7$.

Results and Discussion

Performance profiles are used to compare the performance of optimization models because they allow the illustration of large amounts of instance data[44]. Performance profiles aim to succinctly illustrate the performance of different formulations and can be interpreted by recognizing that every instance is solved by every formulation. Our data is normalized by the formulation that solved the instance the fastest, so the performance curve for a formulation illustrates the ratio of the solution time of that formulation relative to the best formulation time. The y-axis shows the fraction of all instances that are solved faster than the fastest formulation when sped up by a performance multiplier k, which is shown on the x-axis. Alternatively, one can view the performance multiplier k as a value that solution times are divided by when compared against the fastest solution time for an instance, and the y-axis denotes the fraction of all instances solved faster than the fastest solution time after dividing by k. Therefore, every formulation eventually reaches a y-axis value of 1.0 given a sufficiently large multiplier k, but the formulations approaching a y-axis value of 1.0 faster perform better (i.e., the formulations that do not require a large multiplier k to reach the fastest formulation time perform better). This also implies that the starting y-axis value of a formulation at $k = 1$ is the fraction of all instances that a formulation solved the fastest.

To generate the performance profiles, all instances in this work are solved to optimality and sent to a Linux computing cluster (2.8 GHz Intel Cascade Lake processors) with a resource limit of 24 hours and 8 GB of memory. Additionally, GAMS version 36.1 is used with CPLEX 20.1 as the solver, but a brief study was also performed to analyze the impact of RKVs when Gurobi 10.0.0 is used as the solver (see Supporting Information).

Comparison of All RKVs

The first test we conducted evaluates the computational enhancements that implementing various RKVs can yield. We used 96 cost minimization instances that were generated using a combination of three systems, four demand profiles, two processing rate ranges ($\beta_{i,j}^{\text{MIN}}/\beta_{i,j}^{\text{MAX}}$), two run length ranges ($\tau_{i,j}^{\text{MIN}}/\tau_{i,j}^{\text{MAX}}$), and three horizons (see Supporting Information for detailed instance data). We assess the performance of adding each individual RKV to the model as well as many combinations of RKVs. Note that the term "model" is used synonymously with "formulation", but we specifically use the term "reformulation" to refer to any models incorporating RKVs. Table 1 describes the naming convention of the base reformulations, where we consider the addition of only one RKV. In cases where multiple RKVs were added, the letters after the period are appended to the reformulation's name. For example, a model with Eqs. (12)-(14) would be denoted as "X.BIJ" because it incorporates the constraints in the X.B, X.I, and X.J reformulations. In cases where RKVs from both classes (involving both $X_{i,j,n}$ and

$Y^{S}_{i,j,n}$ binary variables) are implemented, a semicolon is used to separate the two classes. For example, a model with Eqs. (15)-(18) would be denoted as "X.NA;Y.BI".

Table 1. Naming convention of reformulations given the RKVs added to the original model (Eqs. (1)–(11)).

Reformulations	Eqs. added	RKVs added
X.B	(12)	$N^{X}_{i,j}$
X.I	(13)	N^{X}_{i}
X.J	(14)	N^{X}_{j}
X.N	(15)	N^{X}_{n}
X.A	(16)	N^{X}
Y.B	(17)	$N^{Y}_{i,j}$
Y.I	(18)	N^{Y}_{i}
Y.J	(19)	N^{Y}_{j}
Y.N	(20)	N^{Y}_{n}
Y.A	(21)	N^{Y}

Table 2 illustrates the number of variables that are added to a model when different RKV reformulations are incorporated. Since the number of some RKVs depend on the number of units, subtasks, or time periods, the overall number of variables being added are instance-specific.

Table 2. Number of variables in the original model and the variables added by RKV reformulations are shown for three select instances (see Supporting Information for instance data).

Model	**System1.d1.e1.h24.t1**	**System2.d4.e2.h36.t2**	**System3.d3.e2.h36.t1**
Original	4726	8622	6957
X.B	13	14	13
X.I	9	10	9
X.J	7	8	7
X.N	25	37	37
X.A	1	1	1
Y.B	13	14	13
Y.I	9	10	9
Y.J	7	8	7
Y.N	25	37	37
Y.A	1	1	1

Figure 4 illustrates the performance profile of the original model and seven other selected reformulations. For clarity, we do not show all of the combinations of RKVs that were tested because 20 different profiles would be difficult to discern from each another (see Supporting Information for all formulation data). The two reformulations that performed the best were Y.I and X.I;Y.I, so the N^{Y}_{i} RKV was present in both of the two best-performing reformulations. Interestingly, the reformulation with all RKVs (X.BIJNA;Y.BIJNA) performed better than the original model but not as well as some of the other reformulations with fewer RKVs. This implies that there is a tradeoff between providing the solver with the ability to branch on RKVs to reduce solution times and increasing the number of

variables in the model. For example, when N_n^{X} and N_n^{Y} were removed from X.BIJNA;Y.BIJNA, resulting in X.BIJA;Y.BIJA, the performance of the reformulation improved.

We observe that the inclusion of N_i^{Y} has a substantial impact on reducing solution times considering that Y.I has the fastest solution time for over 40% of all instances, and the second fastest reformulation, X.I;Y.I, which has the fastest solution time for about 22% of all instances, also contains N_i^{Y}. This is an interesting observation because X.I does not perform much differently from the original model implying that the vast majority of solution time improvements exhibited by the X.I;Y.I reformulation exclusively came from N_i^{Y}. Much of the solution time improvements other reformulations exhibited can also be attributed to the addition of N_i^{Y}. It is clear that incorporating various RKVs into the model can yield significant computation time improvements, but the most noteworthy improvements come from the addition of Eq. (18).

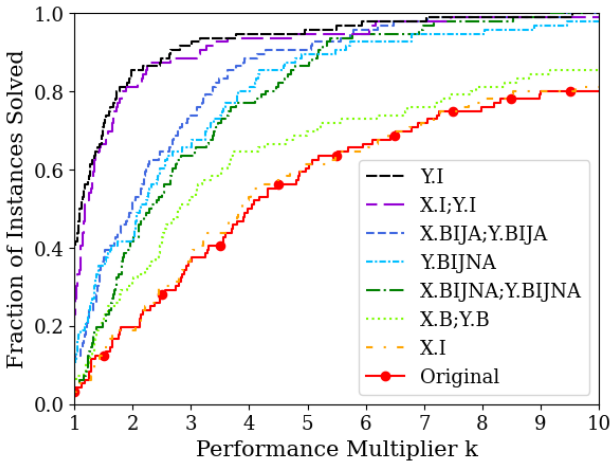


Figure 4. The performance profiles of eight formulations are illustrated. All reformulations that incorporate the RKVs involving $Y_{i,j,n}^{S}$ perform significantly better than the original model (red).

When we consider the implication of what the N_i^{Y} RKV represents, it makes intuitive sense that branching on a variable which tracks the number of runs will likely result in a significantly different objective function value. Inversely, X.N and Y.N, two of the poorer performing reformulations, contain RKVs that track the number of subtasks and runs executed at a time point n, respectively. Branching on such variables is not likely to lead to a significant change (if any) in the objective function value because branching on the number of subtasks/runs executed at time point n could result in changes that could be offset by a change in the number of subtasks/runs executed at time point $n+1$, yielding a solution with a similar (or, in some cases, the exact same) objective function value. This is exemplified when comparing the reformulation with all RKVs (X.BIJNA;Y.BIJNA) to the reformulation with all RKVs except for N_n^{X} and N_n^{Y} (X.BIJA;Y.BIJA) because computational improvements can be seen with the removal of these two RKVs.

Branching Priorities Using Subtask Utilization

Next, we test the impact of prioritizing branching on certain RKVs to determine whether further reductions can be made to computation times. Note that changing branching priorities does not alter a model's constraints or variables but involves modifying the configuration of the solver, so the term "reformulation" is inadequate for distinguishing between these configurations. Thus, we use the term "configuration" to differentiate models that only differ in their branching priorities.

Given the findings in the previous subsection, we focus our branching prioritization efforts on the N_i^{Y} RKVs. Since there is an N_i^{Y} integer variable for every production subtask i, less than a dozen total variables are added to every instance. However, it is difficult to distinguish configurations from each other when all N_i^{Y} RKVs are present in the model and only a single variable's branching priority is being changed. To draw a clearer visual distinction between different configurations, we add all N_i^{Y} RKVs to the model (i.e., reformulation Y.I) and begin by prioritizing branching on one RKV based on certain system attributes. Each successive configuration progressively prioritizes an additional N_i^{Y} RKV until all N_i^{Y} RKVs are eventually prioritized (see Table 3 for example).

Using LP Relaxation to Calculate Utilization

The first system attribute that we use to determine branching priorities is utilization. In order to determine how frequently a subtask is executed, a utilization metric ($Util_i$) is calculated. In this method, the LP relaxation is quickly solved and used to calculate

$$Util_i = \sum_{j \in \mathbf{J}_i} \sum_{n \in \mathbf{N}} \left(X^*_{i,j,n} \tau_{i,j} + \sum_{i' \in \mathbf{I}_i^{\mathrm{TR+}} \cup \mathbf{I}_i^{\mathrm{TR-}}} X^*_{i',j,n} \tau_{i',j} \right), \qquad \forall\, i \in \mathbf{I}^{\mathrm{P}}, \tag{22}$$

where $X^*_{i,j,n}$ is the solution of the LP relaxed problem. Note that the processing time of transitioning to and from a production subtask is also considered; however, because N_i^{Y} is written for production subtasks, we correspondingly calculate $Utili_i$ for every production subtask. The naming convention of configurations with branching prioritization based on $Utili_i$ are prefixed with "LPR" to denote that the LP relaxation is used to determine branching priorities. Additionally, "M" is added to the configuration names to denote that the prioritization order starts with RKVs associated with the most utilized tasks (see Table 3).

Figure 5 illustrates the impact of prioritizing RKVs based on $Utili_i$, and the original model (red) with no RKVs is shown for comparison. Configuration LPR.M.I0 does not prioritize any RKVs, but LPR.M.I1 prioritizes branching on the RKV associated with the most utilized task (based on the highest value of $Util_i$). For configuration LPR.M.I2, we prioritize branching on the RKVs associated with the two most utilized tasks; the RKV associated with the most utilized task is given the highest priority, and the RKV associated with the second most utilized task is given the second highest priority. This

pattern is continued until all N_i^{Y} RKVs are prioritized in the order of the most to least utilized task for configuration LPR.M.I9, as shown in Table 3. Note that only branching on the N_i^{Y} RKVs is prioritized, that is, the $X_{i,j,n}$ and $Y_{i,j,n}^{\mathrm{S}}$ binary variables are not prioritized.

Table 3. Naming convention for the "LPR.M" configurations. Index $i1$ indicates the most utilized production subtask, $i2$ the second most utilized production subtask, etc. Ordinal numbers signify branching priorities: "1st" denotes highest priority, "2nd" denotes second highest priority, etc.

Configuration	Branching priorities
LPR.M.I0	None
LPR.M.I1	1st: N_{i1}^{Y}
LPR.M.I2	1st: N_{i1}^{Y}, 2nd: N_{i2}^{Y}
LPR.M.I3	1st: N_{i1}^{Y}, 2nd: N_{i2}^{Y}, 3rd: N_{i3}^{Y}
…	…
LPR.M.I9	All N_i^{Y} RKVs prioritized from N_{i1}^{Y} (1st) to N_{i9}^{Y} (9th)

All configurations with some RKVs prioritized outperform the configuration with no branching priorities (LPR.M.I0), and LPR.M.I9, which prioritizes all RKVs from most to least subtask utilization, performs the best, solving over 50% of all instances the fastest.

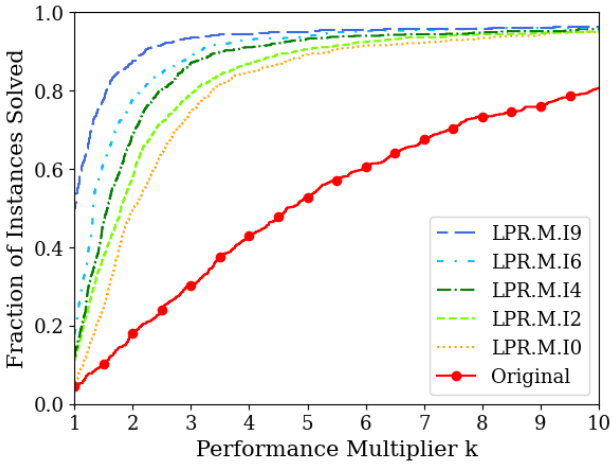


Figure 5. The performance profiles of the original model (red) and five configurations of the Y.I reformulation. Naming conventions given in Table 3. Some configurations are not depicted for clarity.

Comparing Highest and Lowest Utilization

Although Figure 5 illustrates that prioritizing branching on N_i^{Y} can reduce solution times, the decision to prioritize RKVs starting from the most to least utilized subtask is based on the intuition that branching on variables for more utilized subtasks (bottlenecks) plays a larger role in closing the optimality gap. To further study the impact of prioritizing RKVs based on subtask utilization, the same experiment is run with the order of RKV prioritization reversed. The prefix "L" denotes prioritizing from least to most utilization. The RKV corresponding to the least utilized subtask is prioritized in configuration LPR.L.I1, the RKVs corresponding to the two least utilized subtasks are

prioritized in configuration LPR.L.I2, etc. The prioritization is also ordered from the RKV associated with the least utilized subtask to the RKVs associated with more utilized subtasks (see Table 4).

The "LPR.M" and "LPR.L" configurations are compared in Figure 6. The LPR.L.I0 and LPR.M.I0 configurations can be used as controls because they are the exact same; they contain all N_i^{Y} RKVs but do not implement any branching priorities. Interestingly, the "LPR.L" configurations perform better than their "LPR.M" counterparts. For example, LPR.L.I2 (dark green) performs better than LPR.M.I2 (light green). The same can be seen with LPR.L.I6 (light purple), which is one of the best performing configurations, while its counterpart, LPR.M.I6 (dark blue), performs significantly worse. We theorize that subtasks with a lower utilization could play an important role in closing the optimality gap because there is more flexibility in the timing of their operations. Meanwhile, subtasks with high utilization (bottlenecks) have less operational flexibility, so there are fewer feasible schedules for the solver to investigate when fixing the associated decision variables. Thus, prioritizing branching on RKVs associated with less utilized subtasks yields larger solution time reductions.

Table 4. Naming convention for the "LPR.L" configurations. Index $i1$ indicates the most utilized production subtask, $i2$ the second most utilized production subtask, etc. Ordinal numbers signify branching priorities: "1st" denotes highest priority, "2nd" denotes second highest priority, etc.

Configuration	Branching priorities
LPR.L.I0	None
LPR.L.I1	1st: N_{i9}^{Y}
LPR.L.I2	1st: N_{i9}^{Y}, 2nd: N_{i8}^{Y}
LPR.L.I3	1st: N_{i9}^{Y}, 2nd: N_{i8}^{Y}, 3rd: N_{i7}^{Y}
…	…
LPR.L.I9	All N_i^{Y} RKVs prioritized from N_{i9}^{Y} (1st) to N_{i1}^{Y} (9th)

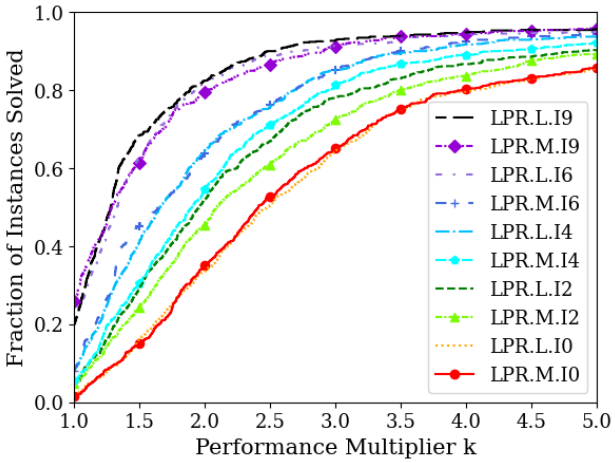


Figure 6. The performance profiles of 10 configurations of the Y.I reformulation. Naming conventions given in Tables 3 and 4. Some configurations are not depicted for clarity.

Studying Equal Prioritization

We have only considered distinct branching prioritization for each RKV, that is, if a subtask has the lowest utilization, its associated RKV is assigned the highest priority relative to all other RKVs. Similarly, the RKV associated with the second-lowest subtask utilization is assigned the second-highest prioritization, and this pattern continues accordingly. Here, we shift our focus to investigating whether equal prioritization of RKVs has a more preferable impact on solution times (see Table 5). Given our previous findings, we continue to start our prioritization efforts on RKVs associated with the least to most utilized subtask. In Figure 7, the prefix "E" denotes that the prioritized RKVs have the same priorities. Configurations range from I0 to I9 with the details given in Table 5. Once again, LPR.E.I0 and LPR.L.I0 can be thought of as controls because they are the same configuration with no prioritizations. There does not appear to be a noticeable difference between the two prioritization methods when fewer RKVs are prioritized (I0, I2, and I4). However, when more than five RKVs are prioritized (I6 and I9), a distinction can be seen where the configurations that prioritize RKVs equally begin plateauing in performance. Configurations LPR.E.I6 and LPR.E.I9 perform similarly, as do LPR.L.I6 and LPR.L.I9, but LPR.L.I9 performs better than LPR.E.I9.

Table 5. Naming convention for the "LPR.E" configurations. Index $i1$ indicates the most utilized production subtask, $i2$ the second most utilized production subtask, etc. Ordinal numbers signify branching priorities: "1st" denotes highest priority, "2nd" denotes second highest priority, etc.

Configuration	Branching priorities
LPR.E.I0	None
LPR.E.I1	1^{st}: N_{i9}^{Y}
LPR.E.I2	1^{st}: N_{i9}^{Y}, 1^{st}: N_{i8}^{Y}
LPR.E.I3	1^{st}: N_{i9}^{Y}, 1^{st}: N_{i8}^{Y}, 1^{st}: N_{i7}^{Y}
…	…
LPR.E.I9	All N_{i}^{Y} RKVs prioritized with equal priorities

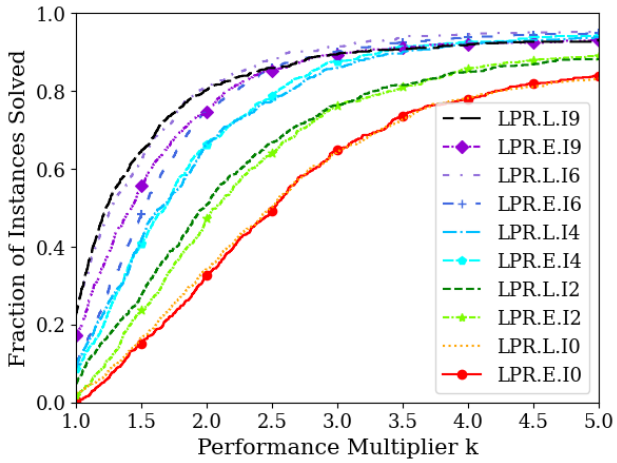


Figure 7. The performance profiles of 10 configurations of the Y.I reformulation. Naming conventions given in Tables 4 and 5. Some configurations are not depicted for clarity.

Preliminary testing (not shown) was performed to determine the effect of implementing branching priorities on RKVs when $X_{i,j,n}$ and $Y^{S}_{i,j,n}$ are not present in the objective function, which is unlikely given that there are inherent costs associated with processing a subtask/run. Our findings show that solution times increase.

Using DPA to Calculate Utilization

Samadi and Maravelias recently proposed a Demand Propagation Algorithm (DPA) to calculate production lower bounds in continuous production scheduling problems[33]. Tightening constraints that use the bounds calculated by the DPA can be implemented to reduce computational resources. However, since the DPA calculates minimum production bounds for each subtask in a system, it, in a sense, determines how utilized a subtask is for a given instance. As such, we investigate whether $Util_i$ yields a better metric for subtask utilization in terms of reducing solution times or if the bound calculated by the DPA is a better metric. We note that the DPA considers conversion coefficients, maximum/minimum processing rates, maximum/minimum task durations, time horizon, and demand profiles in its calculation of minimum production bounds.

The configurations using the DPA method of calculating subtask utilization have the prefix "DPA". Figure 8 compares the performance of the "LPR" to the "DPA" configurations. Similar to before, both I0 configurations contain all N_i^{Y} RKVs with no prioritizations, which means LPR.L.I0 and DPA.L.I0 are the same configuration. Since in the previous subsections we demonstrated that prioritizing branching on RKVs corresponding to the less utilized subtasks benefits solution times more, we continue using this strategy. The naming convention for the "DPA" configurations are shown in Table 6, which is similar to the naming convention shown in Table 4 for the "LPR" configurations.

Though not significantly different, the "LPR" configuration appears to perform slightly better than the "DPA" configurations. Not seeing a large discrepancy between the two methods is expected because both attempt to rank subtasks based on utilization, so they should not be significantly different from each other. From Figure 8, LPR.L.I9 and DPA.L.I9 both prioritize all RKVs in the order of least to most utilized subtask, but LPR.L.I9 performs the best, having the fastest solution time for over a third of instances. In terms of usefully ranking subtask utilization, it appears that the LP relaxation method is a better metric for calculating subtask utilization than the DPA method.

Table 6. Naming convention for the "DPA.L" configurations. Index $i1$ indicates the most utilized production subtask, $i2$ the second most utilized production subtask, etc. Ordinal numbers signify branching priorities: "1st" denotes highest priority, "2nd" denotes second highest priority, etc.

Configuration	Branching priorities
DPA.L.I0	None
DPA.L.I1	1st: N_{i9}^{Y}
DPA.L.I2	1st: N_{i9}^{Y}, 2nd: N_{i8}^{Y}
DPA.L.I3	1st: N_{i9}^{Y}, 2nd: N_{i8}^{Y}, 3rd: N_{i7}^{Y}

...		...	
DPA.L.I9		All N_i^{Y} RKVs prioritized from N_{i9}^{Y} (1st) to N_{i1}^{Y} (9th)	

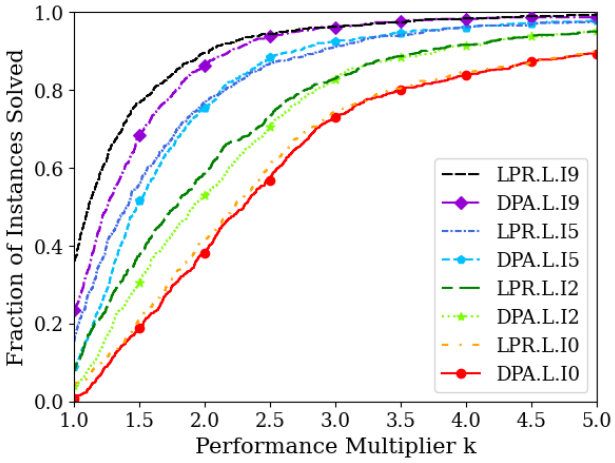


Figure 8. The performance profiles of 8 configurations of the Y.I reformulation. Naming conventions given in Tables 4 and 6. Some configurations are not depicted for clarity.

Branching Priorities Using Unit Demand

We previously considered subtask utilization to determine how impactful branching on specific RKVs is. Another system attribute we investigate is unit "demand" because RKVs associated with units that have a lot of operational flexibility might be more impactful in terms of solution times. The method we use to calculate the demand of a unit (Dem_j) is by first solving the LP relaxation of an instance and then summing the dual variable ($Cliq_{j,n}^*$) of the unit's clique constraint (Eq. (1)) over all time points n:

$$Dem_j = \sum_{n \in \mathbf{N}} Cliq_{j,n}^*, \qquad \forall\, j \in \mathbf{J}. \tag{23}$$

Note that Dem_j is calculated for every unit, but N_i^{Y} is written for every production subtask. For this reason, we must link units to the subtasks that they can process in order to implement branching priorities on the N_i^{Y} RKVs. Some units can process multiple subtasks, so subtasks that correspond to the same unit are all equally prioritized based on the unit's calculated Dem_j. If multiple units have the same Dem_j value, then all associated RKVs are equally prioritized.

Figure 9 illustrates the performance of several configurations ("DEM") with branching priorities based on Dem_j calculated using Eq. (23). Naming conventions for these configurations are shown in Table 7. Ultimately, DEM.L.I9 and DEM.M.I9 prioritize all N_i^{Y} RKVs, but DEM.L.I9 gives the RKVs associated with units that have the least demand a higher branching priority while DEM.M.I9 gives the RKVs associated with units that have the highest Dem_j a higher branching priority.

Table 7. Naming convention for the "DEM.M" and "DEM.L" configurations. Index $i1$ indicates the production subtask associated with the highest Dem_j unit, etc. Ordinal numbers signify branching priorities: "1st" denotes highest priority, "2nd" denotes second highest priority, etc.

Configuration	Branching priorities
DEM.M.I0	None
DEM.M.I1	1st: N_{i1}^{Y}
DEM.M.I2	1st: N_{i1}^{Y}, 2nd: N_{i2}^{Y}
DEM.M.I3	1st: N_{i1}^{Y}, 2nd: N_{i2}^{Y}, 3rd: N_{i3}^{Y}
…	…
DEM.M.I9	All N_{i}^{Y} RKVs prioritized from N_{i1}^{Y} (1st) to N_{i9}^{Y} (9th)
DEM.L.I0	None
DEM.L.I1	1st: N_{i9}^{Y}
DEM.L.I2	1st: N_{i9}^{Y}, 2nd: N_{i8}^{Y}
DEM.L.I3	1st: N_{i9}^{Y}, 2nd: N_{i8}^{Y}, 3rd: N_{i7}^{Y}
…	…
DEM.L.I9	All N_{i}^{Y} RKVs prioritized from N_{i9}^{Y} (1st) to N_{i1}^{Y} (9th)

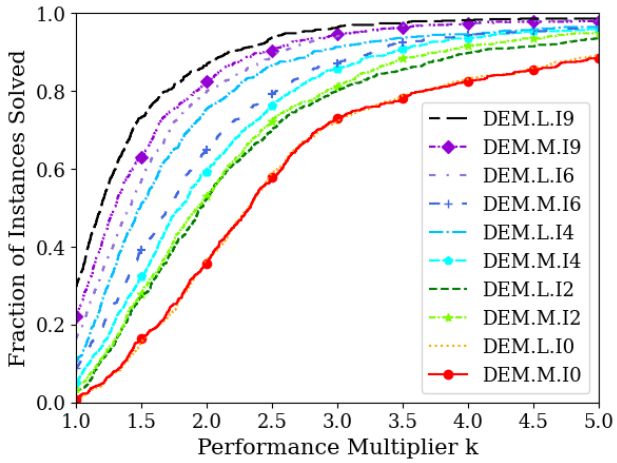


Figure 9. The performance profiles of 10 configurations of the Y.I reformulation. Naming conventions given in Table 7. Some configurations are not depicted for clarity.

Prioritizing branching on RKVs associated with lower Dem_j units first yields better computational results. For example, DEM.L.I6 (light purple) performs better than DEM.M.I6 (dark blue), and DEM.L.I4 (light blue) performs better than DEM.M.I4 (cyan). This is likely due to the same reasons highlighted for Figure 6: when the solver focuses on branching on the RKVs associated with lower Dem_j units (i.e., more operational flexibility), the optimality gap can be reduced more quickly.

Comparison of All Prioritization Methods

Finally, we compare the best configurations from the previous subsections: LPR.L.I9 (from the LP relaxation calculation of subtask utilization), DPA.L.I9 (from the DPA calculation of subtask utilization), and DEM.L.I9 (from the unit demand calculation). The results of these three configurations, alongside the original model and the Y.I reformulation, are illustrated in Figure 10.

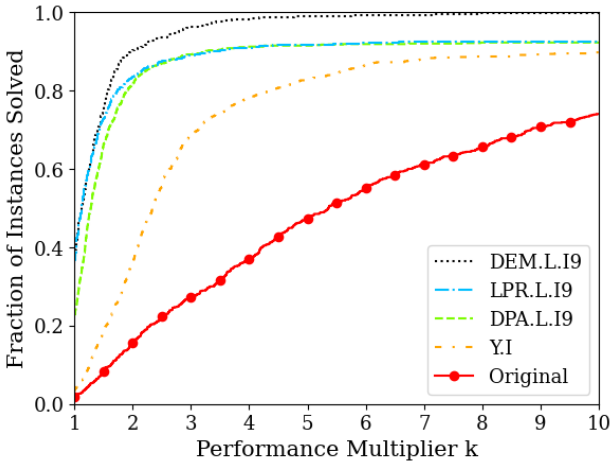


Figure 10. The performance profiles of the Y.I reformulation, three configurations of the Y.I reformulation, and the original model.

The two methods that rely on calculating subtask utilization (LPR.L.I9 and DPA.L.I9) perform similarly relative to the other profiles, but all three configurations with branching priorities perform significantly better than the Y.I reformulation without any branching priorities. The best-performing configuration was DEM.L.I9. When comparing with the original model, the DEM.L.I9 configuration reduces the solution times of about 25% of instances by over an order of magnitude, and average solution times for DEM.L.I9 are 5.5 times faster than the original model.

We conjecture that the "DEM" configurations reduce solution times better than the other configurations because several units are not bottlenecks (or heavily utilized), which yields $Dem_j = 0$ for these units, so their associated RKVs are equally prioritized. We hypothesize that this gives the solver some additional branching flexibility (i.e., the option to select which of the equally prioritized RKVs to branch on). This ultimately results in a combination of prioritizing branching on some RKVs while not over-specifying priorities, which will strictly require the solver to focus branching on specific RKVs in the order we determined. The flexibility that the "DEM" configurations afford the solver are likely the cause of the computational improvements.

Conclusions

This work focuses on addressing the computational challenges inherent to production scheduling. Specifically, we introduce RKVs into MILP models for general continuous production scheduling problems. We show that the proposed reformulations, employing the new RKVs, are significantly more efficient than the original models. Moreover, our results provide insights on how prioritizing branching on RKVs, relative to other binary variables, offers further computational improvements. Our analysis extends to the study of system attributes, such as subtask and unit utilization, to discern the efficiency of prioritizing branching on RKVs associated with specific subtasks or units. Our results

suggest that such prioritization strategies can lead to additional enhancements; the best-performing solver configurations of the Y.I reformulation reduced the solution times of over half of the instances by a factor of 5, and for 25% of the instances by over an order of magnitude. We note that while the above RKVs are implemented in a discrete-time MILP formulation, similar variables can be introduced to continuous-time formulations or even other types of MILP models that have similar structure.

Supporting Information

The "Supporting Information for Publication.docx" file contains information on instance generation, and the "Supporting Information for Publication_Raw Data.xlsx" file contains raw solution time data. This information is available free of charge via the Internet at https://pubs.acs.org.

Acknowledgements

The authors acknowledge financial support from the National Science Foundation under grant CBET-2026980 and NEC Laboratories America, Inc.

Nomenclature

Sets

$i \in \mathbf{I}$	subtasks
$j \in \mathbf{J}$	units
$k \in \mathbf{K}$	materials
$n \in \mathbf{N}$	time points/periods
$t \in \mathbf{T}$	time points/periods relative to the start of a subtask

Subsets

$\mathbf{I}_j$	subtasks that can be processed by unit j
$\mathbf{I}_k^+ / \mathbf{I}_k^-$	subtasks that produce/consume material k
$\mathbf{I}^{\mathrm{DT}}$	production subtasks associated with direct transitions
$\mathbf{I}_i^{\mathrm{DT}+} / \mathbf{I}_i^{\mathrm{DT}-}$	direct transitions to/from production subtask i
$\mathbf{I}^{\mathrm{P}}$	production subtasks
$\mathbf{I}_i^{\mathrm{SD}}$	shutdowns associated with production subtask i

$\mathbf{I}^{\mathrm{SS}}$ production subtasks associated with startups or shutdowns

$\mathbf{I}_i^{\mathrm{SU}}$ startups associated with production subtask i

$\mathbf{I}_i^{\mathrm{TR+}}/\mathbf{I}_i^{\mathrm{TR-}}$ transitions to/from production subtask i

$\mathbf{J}_i$ units that can process subtask i

$\mathbf{J}^{\mathrm{SS}}$ units associated with startups or shutdowns

Parameters

$\alpha_{i,j}^{\mathrm{MIN}}$ minimum time needed to transition to and from a run

$\beta_{i,j}^{\mathrm{MIN}}/\beta_{i,j}^{\mathrm{MAX}}$ minimum/maximum processing rate

$\gamma_{i,j,n}^{\mathrm{B}}/\gamma_{i,j,n}^{\mathrm{X}}$ variable/fixed subtask execution cost

$\gamma_{i,j,n}^{\mathrm{Y}}$ run-starting cost

γ_k^{S} inventory cost

$\xi_{k,n}$ material deliveries (>0) or orders (<0)

$\rho_{i,k,t}$ conversion coefficient

$\tau_{i,j}$ processing time

$\tau_{i,j}^{\mathrm{MIN}}/\tau_{i,j}^{\mathrm{MAX}}$ minimum/maximum run length

$\chi_k^{\mathrm{MIN}}/\chi_k^{\mathrm{MAX}}$ minimum/maximum inventory capacity

Nonnegative Continuous Variables

$B_{i,j,n}$ processing rate of subtask i in unit j starting at time point n

$S_{k,n}$ inventory level of material k during time period n

Binary Variables

$X_{i,j,n}$ =1 if subtask i is processed in unit j staring at time point n

$\hat{X}_{j,n}^{\mathrm{I}}$ =1 if unit j is idle during time period n

$Y_{i,j,n}^{\mathrm{S}}/Y_{i,j,n}^{\mathrm{E}}$ =1 if a run of continuous task i in unit j starts/ends at time point n

Record Keeping Variables

$N_{i,j}^{\mathrm{X}}$ sum of $X_{i,j,n}$ over all time points n

N_i^{X} sum of $X_{i,j,n}$ over all units j and time points n

N_j^{X} sum of $X_{i,j,n}$ over all subtasks i and time points n

N_n^{X} sum of $X_{i,j,n}$ over all subtasks i and units j

N^{X} sum of $X_{i,j,n}$ over all subtasks i, units j, and time points n

$N_{i,j}^{\mathrm{Y}}$ sum of $Y_{i,j,n}^{S}$ over all time points n

N_i^{Y} sum of $Y_{i,j,n}^{S}$ over all units j and time points n

N_j^{Y} sum of $Y_{i,j,n}^{S}$ over all subtasks i and time points n

N_n^{Y} sum of $Y_{i,j,n}^{S}$ over all subtasks i and units j

N^{Y} sum of $Y_{i,j,n}^{S}$ over all subtasks i, units j, and time points n

References

1. Subrahmanyam S, Bassett MH, Pekny JF, Reklaitis G V. Issues in solving large scale planning, design and scheduling problems in batch chemical plants. *Comput Chem Eng*. 1995;19(SUPPL. 1):577-582. doi:10.1016/0098-1354(95)87097-0

2. Harjunkoski I, Maravelias CT, Bongers P, et al. Scope for industrial applications of production scheduling models and solution methods. *Comput Chem Eng*. 2014;62:161-193. doi:10.1016/J.COMPCHEMENG.2013.12.001

3. Georgiadis GP, Elekidis AP, Georgiadis MC. Optimization-based scheduling for the process industries: From theory to real-life industrial applications. *Processes*. 2019;7(7). doi:10.3390/pr7070438

4. Kelly JD, Zyngier D. Unit-operation nonlinear modeling for planning and scheduling applications. *Optim Eng*. 2017;18(1):133-154. doi:10.1007/S11081-016-9312-7/FIGURES/7

5. Zyngier D, Kelly JD. *Optimization and Logistics Challenges in the Enterprise*. Springer; 2009.

6. Castro PM, Westerlund J, Forssell S. Scheduling of a continuous plant with recycling of byproducts: A case study from a tissue paper mill. *Comput Chem Eng*. 2009;33(1):347-358. doi:10.1016/j.compchemeng.2008.10.004

7. Ku HM, Karimi IA. Scheduling in serial multiproduct batch processes with finite interstage storage: mixed integer linear program formulation. *Ind Eng Chem Res*. 1988;27(10):1840-1848.

8. Mendez CA, Cerda J. An MILP Continuous-Time Framework for Short-Term Scheduling of Multipurpose Batch Processes Under Different Operation Strategies. *Optim Eng*. 2003;4:7-22.

9. Kondili E, Pantelides CC, Sargent RWH. A general algorithm for short-term scheduling of batch operations-I. MILP formulation. *Comput Chem Eng*. 1993;17(2):211-227. doi:10.1016/0098-1354(93)80015-F

10. Wolsey LA. MIP modelling of changeovers in production planning and scheduling problems. *Eur J Oper Res*. 1997;99:154-165.

11. Cafaro DC, Grossmann IE. Strengthening discrete-time scheduling formulations by introducing the concept of campaigns. *Comput Chem Eng*. 2020;143:107101. doi:10.1016/J.COMPCHEMENG.2020.107101

12. Velez S, Dong Y, Maravelias CT. Changeover formulations for discrete-time mixed-integer programming scheduling models. *Eur J Oper Res*. 2017;260(3):949-963. doi:10.1016/j.ejor.2017.01.004

13. Kelly JD, Zyngier D. An improved MILP modeling of sequence-dependent switchovers for discrete-time scheduling problems. *Ind Eng Chem Res*. 2007;46(14):4964-4973. doi:10.1021/IE061572G/ASSET/IMAGES/LARGE/IE061572GF00007.JPEG

14. Basán NP, Grossmann IE, Gopalakrishnan A, Lotero I, Méndez CA. Novel MILP Scheduling Model for Power-Intensive Processes under Time-Sensitive Electricity Prices. *Ind Eng Chem Res*. 2018;57(5):1581-1592. doi:10.1021/ACS.IECR.7B04435/SUPPL_FILE/IE7B04435_SI_001.PDF

15. Pattison RC, Touretzky CR, Johansson T, Harjunkoski I, Baldea M. Optimal Process Operations in Fast-Changing Electricity Markets: Framework for Scheduling with Low-Order Dynamic Models and an Air Separation Application. *Ind Eng Chem Res*. 2016;55(16):4562-4584.

doi:10.1021/acs.iecr.5b03499

16. Mitra S, Grossmann IE, Pinto JM, Arora N. Optimal production planning under time-sensitive electricity prices for continuous power-intensive processes. *Comput Chem Eng*. 2012;38:171-184. doi:10.1016/J.COMPCHEMENG.2011.09.019

17. Ferris MC, Maravelias CT, Sundaramoorthy A. Simultaneous batching and scheduling using dynamic decomposition on a grid. *INFORMS J Comput*. 2009;21(3):398-410. doi:10.1287/ijoc.1090.0339

18. Giménez DM, Henning GP, Maravelias CT. A novel network-based continuous-time representation for process scheduling: Part I. Main concepts and mathematical formulation. *Comput Chem Eng*. 2009;33(9):1511-1528. doi:10.1016/j.compchemeng.2009.03.007

19. Shaik MA, Floudas CA, Kallrath J, Pitz HJ. Production scheduling of a large-scale industrial continuous plant: Short-term and medium-term scheduling. *Comput Chem Eng*. 2009;33(3):670-686. doi:10.1016/j.compchemeng.2008.08.013

20. Gupta D, Maravelias CT, Wassick JM. From rescheduling to online scheduling. *Chem Eng Res Des*. 2016;116:83-97. doi:10.1016/J.CHERD.2016.10.035

21. McAllister RD, Rawlings JB, Maravelias CT. The inherent robustness of closed-loop scheduling. *Comput Chem Eng*. 2022;159:107678. doi:10.1016/J.COMPCHEMENG.2022.107678

22. Bassett MH, Pekny JF, Reklaitis G V. Decomposition Techniques for the Solution of Large-Scale Scheduling Problems. *AIChE J*. 1996;42(12):3373-3387.

23. Kopanos GM, Méndez CA, 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(2):644-655. doi:10.1016/J.EJOR.2010.06.002

24. Velez S, Maravelias CT. Reformulations and branching methods for mixed-integer programming chemical production scheduling models. *Ind Eng Chem Res*. 2013;52(10):3832-3841. doi:10.1021/ie303421h

25. Merchan AF, Maravelias CT. Reformulations of mixed-integer programming continuous-time models for chemical production scheduling. *Ind Eng Chem Res*. 2014;53(24):10155-10165. doi:10.1021/ie404274b

26. Sahinidis N V., Grossmann IE. Reformulation of multiperiod MILP models for planning and scheduling of chemical processes. *Comput Chem Eng*. 1991;15(4):255-272. doi:10.1016/0098-1354(91)85012-J

27. Yee KL, Shah N. Improving the efficiency of discrete time scheduling formulation. *Comput Chem Eng*. 1998;22(SUPPL.1):S403-S410. doi:10.1016/S0098-1354(98)00081-7

28. Papageorgiou LG, Pantelides CC. Optimal campaign planning/scheduling of multipurpose batch/semicontinuous plants. 2. A mathematical decomposition approach. *Ind Eng Chem Res*. 1996;35(2):510-529. doi:10.1021/IE950082D/ASSET/IMAGES/LARGE/IE950082DF00020.JPEG

29. Subrahmanyam S, Kudva GK, Bassett MH, Pekny JF. Application of Plant Distributed Design and Computing to Batch Scheduling. *AIChE J*. 1996;42(6):1648-1661.

30. Velez S, Maravelias CT. A branch-and-bound algorithm for the solution of chemical production scheduling MIP models using parallel computing. *Comput Chem Eng*. 2013;55:28-39. doi:10.1016/j.compchemeng.2013.03.030

31. Burkard RE, Hatzl J. Review, extensions and computational comparison of MILP formulations for scheduling of batch processes. *Comput Chem Eng*. 2005;29(8):1752-1769. doi:10.1016/j.compchemeng.2005.02.037

32. Janak SL, Floudas CA. Improving unit-specific event based continuous-time approaches for batch processes: Integrality gap and task splitting. *Comput Chem Eng*. 2008;32(4-5):913-955. doi:10.1016/j.compchemeng.2007.03.019

33. Samadi A, Maravelias CT. Computational enhancements of continuous production scheduling MILPs using tightening constraints. *Comput Chem Eng*. 2024;184:108609. doi:10.1016/J.COMPCHEMENG.2024.108609

34. Adelgren N, Maravelias CT. On the utility of production scheduling formulations including record keeping variables. *Comput Ind Eng*. 2023;181(April):109330. doi:10.1016/j.cie.2023.109330

35. Maravelias CT. *Chemical Production Scheduling*. Cambridge University Press; 2021. doi:10.1017/9781316650998

36. Pantelides CC. Unified frameworks for optimal process planning and scheduling. In: *Proceedings on the Second Conference on Foundations of Computer Aided Operations*. CACHE Publications; 1994:253-274.

37. Samadi A, Maravelias CT. A Comprehensive Chemical Production Scheduling Representation. *Comput Chem Eng*. Published online February 1, 2024:108552. doi:10.1016/J.COMPCHEMENG.2023.108552

38. Wu Y, Maravelias CT. A general framework and optimization models for the scheduling of continuous chemical processes. *AIChE J*. 2021;67(10):1-15. doi:10.1002/aic.17344

39. Sundaramoorthy A, Maravelias CT. Computational study of network-based mixed-integer programming approaches for chemical production scheduling. *Ind Eng Chem Res*. 2011;50(9):5023-5040. doi:10.1021/ie101419z

40. Merchan AF, Velez S, Maravelias CT. Tightening methods for continuous-time mixed-integer programming models for chemical production scheduling. *AIChE J*. 2013;59(12):4461-4467. doi:10.1002/aic.14249

41. Floudas CA, Lin X. Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. *Comput Chem Eng*. 2004;28(11):2109-2129. doi:10.1016/J.COMPCHEMENG.2004.05.002

42. Méndez CA, Cerdá J, Grossmann IE, Harjunkoski I, Fahl M. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Comput Chem Eng*. 2006;30(6-7):913-946. doi:10.1016/j.compchemeng.2006.02.008

43. Samadi A, Adelgren N, Maravelias CT. On discrete time chemical production scheduling MILP models containing record keeping variables. *Comput Aided Chem Eng*. 2023;52:433-438. doi:10.1016/B978-0-443-15274-0.50069-X

44. Dolan ED, Moré JJ. Benchmarking optimization software with performance profiles. *Math Program*. 2002;91(2):201-213. doi:10.1016/S0167-2991(08)65284-2

FOR TABLE OF CONTENTS USE ONLY

Continuous Production Scheduling MILP Formulations Using Record Keeping Variables

Amin Samadi [a], Christos T. Maravelias [a,b]

[a] Department of Chemical and Biological Engineering, Princeton University, Princeton, NJ 08540

[b] Andlinger Center for Energy and the Environment, Princeton University, Princeton, NJ 08540