RECENT ADVANCES IN DISCRETE TIME CHEMICAL PRODUCTION SCHEDULING MILP MODELS

Nathan Adelgren [a,1], Amin Samadi [b], and Christos T. Maravelias [a,b]

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

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

Abstract

In this paper, we present techniques that reduce the computational time required to solve discrete time-based mixed-integer programming models for chemical production scheduling. Methods for both batch processes and continuous processes are presented. The results of several computational tests are included to demonstrate the utility of the novel solution strategies, reformulations, and tightening constraints presented herein.

Keywords

batch processes, continuous processes, discrete time models, cut generation, tightening, reformulations

Introduction

In the context of a processing facility that converts raw materials into higher value products by carrying out tasks in processing units, chemical production scheduling involves assigning tasks to units and establishing the times at which the processing of these tasks should begin, possibly subject to a set of restrictions. As these task assignments and timing decisions ultimately determine the overall performance of the plant, scheduling is a crucial decision-making step in chemical manufacturing facilities, with applications in a broad range of systems, from batch production of low-volume products such as pharmaceuticals (Papavasileiou et al., 2007) to crude oil blending (Castro and Grossmann, 2014). Determining an appropriate schedule requires knowledge of the processing times for each task, limitations on the units to which a given task can be assigned, network connectivities, conversion coefficients, and many other complex, application-specific details. Research in the field of chemical production scheduling has aimed to account for various process characteristics such as constraints on utilities and resources (Méndez et al., 2001), changeovers (Wolsey, 1997), and material transfer restrictions (Giménez et al., 2009). The overarching goal of considering this wide array of process characteristics is to develop highly general models that can be readily employed in a range of industrial applications.

Alongside problem generality, another challenge researchers have been addressing is the computational efficiency of scheduling models. Research efforts to reduce the computational costs of chemical production scheduling models have included the development of reformulations (Velez and Maravelias, 2013), decomposition-based algorithms (Harjunkoski and Grossmann, 2002), parallel computing tools (Subrahmanyam et al., 1996), and tightening methods using valid inequalities (Velez et al., 2013).

In this work we present recent advances in discrete time-based mixed-integer linear programming (MILP) modeling strategies for production scheduling. We divide our discussion into two categories: (i) developments for batch processes, and (ii) developments for continuous processes.

Preliminaries

There are significant modeling differences between batch and continuous processes, primarily due to the fact that continuous processes produce (output) and consume (input) materials continuously and simultaneously, whereas batch processes are assumed to consume all required materials at the start of a task and produce all outputs at the end. The amount of material produced/consumed in continuous production depends on both the rate and duration of task execution, where the duration is not strictly fixed. This implies a different number of degrees of freedom (DOF). Batch processes have one DOF, which is simply the batch size, but continuous processes have two DOF: the rate and the duration of time that the task is processed. To account for this discrepancy, MILP models built for batch processes must be modified in order to appropriately model continuous processes. Hence, in this

[1] Corresponding author. Email: na4592@princeton.edu.

work we first discuss advances in MILP modeling techniques designed for batch process and later discuss advances for continuous processes.

We employ the following convention for notation: (i) sets are indicated using bold, upper-case, Roman letters, (ii) indices are indicated using lower-case Roman letters, (iii) parameters are indicated using lower-case Greek letters, and (iv) variables are indicated using non-bold, upper-case, Roman letters. Throughout the work we utilize $i \in \mathbf{I}$ to represent tasks, $j \in \mathbf{J}$ to represent processing units, $k \in \mathbf{K}$ to represent materials, and $n \in \mathbf{N}$ to represent discrete time points. We note that, while the typical use of a $n \in \mathbf{N}$ is to represent a discrete time *point* that is $n\delta$ time units beyond the start of the scheduling horizon, it can also be used to represent the time *period* $[(n-1)\delta, n\delta)$. For a given $i \in \mathbf{I}$ we use $\mathbf{J}_i \subseteq \mathbf{J}$ to represent the set of units capable of processing task i. Additionally, for a given $k \in \mathbf{K}$ we use $\mathbf{I}_k^+, \mathbf{I}_k^- \subseteq \mathbf{I}$ to represent the sets of tasks producing and consuming material k, respectively. The MILP models we employ also rely on the parameters: η — the scheduling horizon; $\delta = \frac{\eta}{|\mathbf{N}|}$ — the discretization time step; $\tau_{i,j}$ — the time required to process task $i \in \mathbf{I}$ in unit $j \in \mathbf{J}_i$. $\beta_j^{\text{MAX}}/\beta_j^{\text{MIN}}$ — the maximum/minimum capacity of unit $j \in \mathbf{J}$; χ_k^{MAX} — the maximum amount of material $k \in \mathbf{K}$ that can be stored; $\rho_{i,k}$ — the conversion coefficient of material $k \in \mathbf{K}$ produced or consumed by task $i \in \mathbf{I}$; and $\xi_{k,n}$ — the net shipment of material $k \in \mathbf{K}$ at time $n \in \mathbf{N}$. The binary variable $X_{i,j,n} \in \{0,1\}$, which equals 1 if task $i \in \mathbf{I}$ begins in unit $j \in \mathbf{J}_i$ at time point $n \in \mathbf{N}$, is also used in our MILP models.

Batch Processes

This section is divided into two parts. In the first we discuss advances for batch processes in single-stage environments, and in the second we discuss advances for batch processes in network environments.

Single-Stage Environments

In recent years both MILP modeling and constraint programming (CP) have emerged as particularly powerful tools for solving single-stage scheduling problems, but both have difficulties solving rather large, challenging problems. Hence, researchers have been developing hybrid MILP/CP approaches capable of capitalizing on the strengths of both methods. To our knowledge, however, all hybrid MILP/CP approaches proposed in the literature rely on a continuous representation of time. Here we present a novel hybrid MILP/CP method that employs a discrete representation of time. Specifically, our proposed approach embeds both a discrete time MILP model and a CP within a branch-and-cut (BC) framework. The discrete time MILP serves as a high level subproblem to which BC is initially applied, and the CP serves as a low level subproblem that is used for checking feasibility and generating cuts throughout the course of the solution process. Our approach works in three phases. First, a pre-processing phase is used to modify the model parameters to ensure that any schedule that can be feasibly implemented in continuous time is also feasible for the discrete time MILP. BC is then employed to compute the discrete time solution having the lowest processing cost. In this phase, each time an integer feasible solution is found we check the continuous time feasibility of the solution and add appropriate cuts in the case of infeasibility. Finally, a post-processing phase is used to convert the task start times assigned by the optimal discrete time solution to feasible values in continuous time.

For the sake of space, we do not include the formulations of the MILP or CP models that we employ, but we note that the MILP model is taken from Chapter 4.4 of (Maravelias, 2021). Additionally, due to the complex nature of the overall procedure we propose, we do not include details of the pre- or post-processing phases. We note, however, that the post-processing phase is equivalent to the second phase of the Discrete Continuous Algorithm (DCA) of Merchan et al. (2016). A high level description of the main step of our proposed approach, namely the processing of integer feasible solutions discovered during the course of BC, is given in Algorithm 1.

Algorithm 1 PROCESSINTEGERSOLUTION(X)

Input: X — an integer feasible solution
Output: An integer feasible solution X^* such that $\mathtt{seq}_j(X^*)$ is feasible for all $j \in \mathbf{J}$, if one has been discovered.

Set $AllFeas = \mathtt{true}$ and $X^* = X$.
for $j \in \mathbf{J}$ **do**
  if $\mathtt{seq}_j(X)$ is not feasible **then**
    Cut off infeasible subsequences of $\mathtt{seq}_j(X)$.
    Let X'_j be the solution to the low-level CP.
    if X'_j is not feasible **then**
      Cut off $\mathtt{asn}_j(X)$.
      Set $AllFeas = \mathtt{false}$.
    else set $\mathtt{seq}_j(X^*) = \mathtt{seq}_j(X'_j)$
  else set $\mathtt{seq}_j(X^*) = \mathtt{seq}_j(X)$.
if $AllFeas$ is $\mathtt{true}$ **then** return X^*.
else return $\emptyset$.

Note that in Algorithm 1, for a given integer feasible solution X and $j \in \mathbf{J}$ we use $\mathtt{asn}_j(X)$ to denote the set of tasks that X assigns to j. Similarly, we use $\mathtt{seq}_j(X)$ to denote the sequence in which the tasks in $\mathtt{asn}_j(X)$ are processed.

Using different pre-processing approaches, we employ three variants of our proposed procedure (denoted DTR full, DTR MIP_0, and RPTT) and compare against three alternative approaches: (i) directly solving the continuous time MILP model described in Chapter 4.3 of Maravelias (2021), (ii) directly solving the discrete time MILP model described in Chapter 4.4 of Maravelias (2021), and (iii) using the hybrid MILP/CP method referred to as Algorithm 4 in (Sadykov and Wolsey, 2006). Each of these methods was programmed using the C++ programming language and a combination of the CPLEX 20.1 C API (for MILP modeling) and C++ API (for CP modeling). All of the computational tests described in this work were conducted using a computing cluster running Springdale Linux 8. For our current test, we set a time limit of 5 hours and utilized 49 randomly generated instances. All six approaches successfully solved 41 of these instances

within the time limit, and results for these instances are summarized in the performance profile displayed in Figure 1.

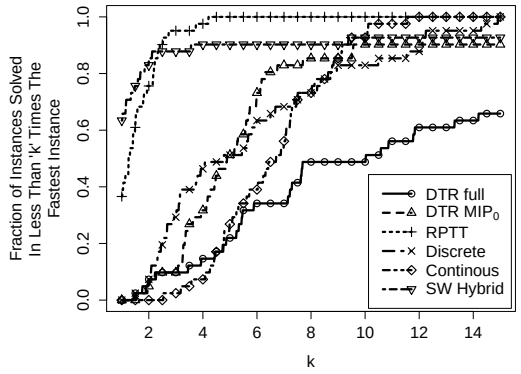

Figure 1: Results for Our Hybrid MILP/CP Approach

Recognize from Figure 1 that while the hybrid MILP/CP method of Sadykov and Wolsey (2006) (SW Hybrid) was the fastest approach for over 60% of the considered instances, our RPTT approach was fastest for the remainder. Although it cannot be ascertained from Figure 1, we point out that the majority of the instances for which the SW Hybrid approach took less time than our RPTT approach were solved very quickly – in most cases both approaches took less than 1 second. Further recognize that for approximately 10% of the considered instances, all other approaches took over 10 times the time utilized by our RPTT approach.

Network Environments

Beginning with the MILP model presented in Chapter 7.2.2 of (Maravelias, 2021), Velez and Maravelias (2013) propose a reformulation that involves adding an integer variable $N_{i,j}$ that represents the number of times task i is carried out in unit j. Specifically, the authors add the constraint

$$\sum_{n\in\mathbf{N}} X_{i,j,n} = N_{i,j} \qquad \forall\, i\in\mathbf{I}, j\in\mathbf{J}_i \qquad (1)$$

and bound $N_{i,j}$ as

$$0 \le N_{i,j} \le \lfloor \eta/\tau_{i,j} \rfloor \qquad \forall\, i\in\mathbf{I}, j\in\mathbf{J}_i. \qquad (2)$$

Velez and Maravelias (2013) state that their motivation for proposing this reformulation is that for many scheduling problems there are multiple feasible solutions having the same objective value and the same (task, unit) pair assignments. Moreover, as these assignments frequently occur at different time points in different solutions, the authors state that branching on $N_{i,j}$ can lead to faster solution times due to the fact that each such branching decision is able to eliminate multiple suboptimal solutions simultaneously.

It can be shown, however, that the benefits of defining $N_{i,j}$ and adding Eqs. (1) and (2) extend beyond those mentioned above because it is not only the branching phase of branch-and-bound (BB) that benefits from these additions. As such, we propose the use of several additional integer variables and associated constraints. Consider the variables

$N_i \in \mathbb{Z}$	the number of times task i is performed,
$N_j \in \mathbb{Z}$	the number of times unit j performs a task,
$N_n \in \mathbb{Z}$	the number of tasks performed at time t, and
$N \in \mathbb{Z}$	the total number of tasks performed,

that can be incorporated using the following constraints and bounds:

$$N_i = \sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} X_{i,j,n}; \quad 0 \le N_i \le \sum_{j\in\mathbf{J}_i} \left\lfloor \frac{\eta}{\tau_{i,j}} \right\rfloor \qquad \forall\, i\in\mathbf{I} \qquad (3)$$

$$N_j = \sum_{i:\, j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} X_{i,j,n}; \quad 0 \le N_j \le \left\lfloor \frac{\eta}{\min\limits_{i:\, j\in\mathbf{J}_i}\{\tau_{i,j}\}} \right\rfloor \forall\, j\in\mathbf{J} \qquad (4)$$

$$N_n = \sum_{i\in\mathbf{I}}\sum_{j\in\mathbf{J}_i} X_{i,j,n}; \quad 0 \le N_n \le \min\{|\mathbf{I}|, |\mathbf{J}|\} \qquad \forall\, n\in\mathbf{N} \qquad (5)$$

$$N = \sum_{i\in\mathbf{I}}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} X_{i,j,n};$$
$$0 \le N \le \min\left\{ \sum_{i\in\mathbf{I}}\sum_{j\in\mathbf{J}_i} \left\lfloor \frac{\eta}{\tau_{i,j}} \right\rfloor, \sum_{j\in\mathbf{J}} \left\lfloor \frac{\eta}{\min\limits_{i:\, j\in\mathbf{J}_i}\{\tau_{i,j}\}} \right\rfloor \right\} \qquad (6)$$

As each of the variables $N_{i,j}$, N_i, N_j, N_t, and N serves to *keep record* of a quantity of interest, we refer to each as a *record keeping variable*. We now summarize the results of a study in which we compare the performance of BB when a carefully chosen subset of these record keeping variables are included within our model. We note that this, as well as the remainder of the tests described in this work, were conducted using GAMS v36.1 for modeling and CPLEX 20.1 as the MILP solver. The tests described here are performed using a combination of instances obtained from `minlp.org` and from the authors of (Velez and Maravelias, 2013). In total, we utilize 15 instances and consider three objectives: (i) makespan minimization, (ii) cost minimization, and (iii) profit maximization. For brevity, and because the results for all objective types displayed relatively similar patterns, we only present results for cost minimization here. We report results for all instances that at least one of the considered formulations was able to solve in under 5 hours. The results are summarized in the performance profile displayed in Figure 2.

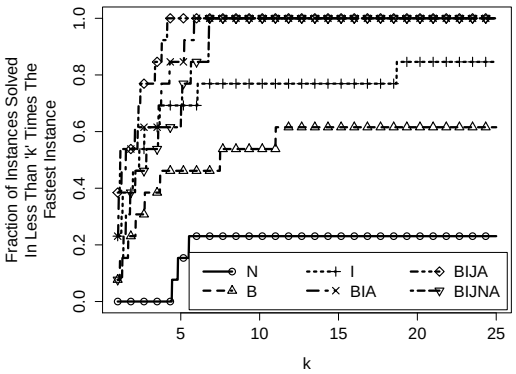

Figure 2: Results for Record Keeping Variables

In Figure 2, we use the following notation: N — no record keeping variables are added to the model; B — only $N_{i,j}$ is added; I — only N_i is added; BIA — $N_{i,j}$, N_i, and N are added simultaneously; BIJA — $N_{i,j}$, N_i, N_j, and N are added simultaneously; and BIJNA — $N_{i,j}$, N_i, N_j, N_n, and N are added simultaneously. We note that N is selected as it represents the default formulation with no record keeping variables included, B is selected as it represents the most promising reformulation considered in (Velez and Maravelias, 2013), and the rest of the considered reformulations are selected due to their relatively strong performance. Of the 15 instances considered, there were 2 that no considered formulation was able to solve in 5 hours, 1 that was solved in under 5 hours by B and BIJA but not by N (B — 2163.8s; BIJA — 16.17s), and 2 that were solved in under 5 hours by BIJA but not be either N or B (BIJA — 1.355s avg). Moreover, of the remaining 10 instances, there were 7 that at least one of the formulations N, B, or BIJA took more than 5 seconds to solve, and on average BIJA solved these in 1.77% of the time used by N and 87.67% of the time used by B.

Continuous Processes

In the field of chemical production scheduling, less focus has been given to continuous processes relative to batch processes because the simultaneous production and consumption of materials alongside the flexibility to choose the duration of task execution adds a layer of processing complexity to the problems. Additionally, batch processes oftentimes have processing time limitations and material handling restrictions, which their continuous process counterparts do not possess, that reduce the feasible region of the problem. Strategies to more efficiently model continuous processes are needed in order to decrease computational times, especially when considering transient operations such as startups, shutdowns, and direct transition tasks that notably escalate model complexity; transient operations can result in the addition of four more binary variables per task in conjunction with the ancillary constraints needed for their modeling. A general optimization framework that accurately represents system dynamics is presented, specifically transient operations. We introduce tightening constraints with the goal of decreasing the computational times of solving continuous production scheduling models.

Modeling Parameters and Variables

A continuous task is modeled using subtasks with processing times of one discrete time step. A run of a continuous task is defined as a set of consecutive, single-period subtasks. The minimum/maximum flowrates of a task $i \in \mathbf{I}$ executed in a unit $j \in \mathbf{J}_i$ during each time period are comparable to batch sizes in the batch process model, so they are represented similarly ($\beta_{i,j}^{\text{MIN}}/\beta_{i,j}^{\text{MAX}}$). In terms of run duration, parameters to enforce run length restrictions ($\tau_{i,j}^{\text{MIN}}/\tau_{i,j}^{\text{MAX}}$) are also applied.

In addition to the $X_{i,j,n}$ binary variables, two new binary variables containing information about the start and end times of a run are employed. Respectively, $Y_{i,j,n}^{\text{S}}$ and $Y_{i,j,n}^{\text{E}}$ signify that a run of task i in unit j starts, or ends, at time n. For details on how these variables are incorporated in the model, see Eqs. (16) and (17) of (Wu and Maravelias, 2021).

Demand Propagation Algorithm

Starting with known customer demands, a preprocessing approach, designated as the demand propagation algorithm (DPA), is used to tighten the original formulation by removing solutions that cannot realistically achieve the desired production demands. The DPA exploits known information such as demand, network structure, conversion coefficients, and processing rates in order to assign values to a set of parameters that we subsequently employ within a novel set of tightening constraints. Specifically, the DPA produces ω_k for each $k \in \mathbf{K}$ and $\tilde{\mu}_i$ for each $i \in \mathbf{I}^{\text{P}}$, where ω_k represents the lower bound on the amount of a material required to satisfy demand and $\tilde{\mu}_i$ represents the lower bound on the production of a task required to satisfy demand.

The DPA uses several record-keeping sets: $\mathbf{K}^{\text{NC}} \subseteq \mathbf{K}$ — the set of materials for which ω_k has not been calculated, $\mathbf{I}^{\text{NC}} \subseteq \mathbf{I}$ — the set of tasks for which $\tilde{\mu}_i$ has not been calculated, $\mathbf{K}^{\text{A}} \subseteq \mathbf{K}$ — the set of materials available for calculating ω_k, and $\mathbf{I}^{\text{A}} \subseteq \mathbf{I}$ — the set of tasks available for calculating $\tilde{\mu}_i$. Also, $\mathbf{K}^{\text{F}} \subseteq \mathbf{K}$ is the set of all final products, and $\mathbf{I}^{\text{P}} \subseteq \mathbf{I}$ is the set of production tasks (i.e., excludes transition tasks).

Algorithm 2 DPA$(\mathbf{I}, \mathbf{J}, \mathbf{K}, \mathbf{N})$

Input: Sets $\mathbf{I}, \mathbf{J}, \mathbf{K}, \mathbf{N}$ (along with all associated parameters)
Output: $\omega_k \ \forall k \in \mathbf{K}$ and $\tilde{\mu}_i \ \forall i \in \mathbf{I}^{\text{P}}$.

Set $\mathbf{K}^{\text{NC}} = \mathbf{K}$, $\mathbf{I}^{\text{NC}} = \mathbf{I}^{\text{P}}$, $\mathbf{K}^{\text{A}} = \mathbf{K}^{\text{F}}$, and $\mathbf{I}^{\text{A}} = \emptyset$.
$\omega_k = \sum_{n \in \mathbf{N}} \xi_{k,n} \qquad \forall k \in \mathbf{K}^{\text{A}}$
while $|\mathbf{I}^{\text{NC}}| + |\mathbf{K}^{\text{NC}}| > 0$ **do**
  $\nu_{i,k} = \max\{\omega_k, 0\} \qquad \forall k \in \mathbf{K}^{\text{A}} : |\mathbf{I}_k^+| = 1, i \in \mathbf{I}_k^+ \bigcap \mathbf{I}^{\text{P}}$
  $\nu_{i,k} = 0 \qquad \forall k \in \mathbf{K}^{\text{A}} : |\mathbf{I}_k^+| > 1, i \in \mathbf{I}_k^+ \bigcap \mathbf{I}^{\text{P}}$
  Set $\mathbf{K}^{\text{NC}} = \mathbf{K}^{\text{NC}} \backslash \mathbf{K}^{\text{A}}$ and $\mathbf{I}^{\text{A}} = \{i : i \in \mathbf{I}_k^+, k \in \mathbf{K}^{\text{A}}\}$.
  Set $\mathbf{K}^{\text{A}} = \emptyset$.
  $\mu_i = \max_{k : i \in \mathbf{I}_k^+} \left\{ \frac{\nu_{i,k}}{\rho_{i,k}} \right\} \qquad \forall i \in \mathbf{I}^{\text{A}}$
  $\tilde{\mu}_i = \mu_i + \Delta\mu_i \qquad \forall i \in \mathbf{I}^{\text{A}}$
  Set $\mathbf{I}^{\text{NC}} = \mathbf{I}^{\text{NC}} \backslash \mathbf{I}^{\text{A}}$ and $\mathbf{K}^{\text{A}} = \{k : i \in \mathbf{I}_k^- \bigcap \mathbf{I}^{\text{A}}\}$.
  Set $\mathbf{I}^{\text{A}} = \emptyset$.
  $\omega_k = -\sum_{i \in \mathbf{I}_k^-} \frac{\tilde{\mu}_i}{\rho_{i,k}} - s_k^0 \qquad \forall k \in \mathbf{K}^{\text{A}}$
return $\left(\{\omega_k : k \in \mathbf{K}\}, \{\tilde{\mu}_i : i \in \mathbf{I}^{\text{P}}\}\right)$

The first step of the DPA is to add all materials to the $\mathbf{K}^{\text{NC}}$ set, all production tasks to the $\mathbf{I}^{\text{NC}}$ set, and all final products to the $\mathbf{K}^{\text{A}}$ set. We also initialize $\mathbf{I}^{\text{A}}$ as the empty set. Then, ω_k is calculated for available materials. The while loop ensures that eventually all production tasks and all materials have an associated $\tilde{\mu}_i$ and ω_k, respectively.

We next calculate $\nu_{i,k}$, a parameter that links materials with the tasks that can produce them, for all available materials $k \in \mathbf{K}^{\text{A}}$ and tasks $i \in \mathbf{I}_k^+$. If more than one task is able to produce the same material, then the minimum processing time of those tasks will be zero because certain tasks could theoretically handle all of the production allowing one task to

not produce. In these cases, a different tightening constraint will need to be used. We then update our record-keeping sets accordingly and calculate μ_i for all available tasks.

On line 8 of Algorithm 2, the term inside the bracket is the total amount of material that all tasks need to produce in order to satisfy demand. However, this quantity might not be in the attainable production range of the task or unit based on operational restrictions such as run length duration, $\tau^{\text{MIN}}_{i,j}/\tau^{\text{MAX}}_{i,j}$, or production rate, $\beta^{\text{MIN}}_{i,j}/\beta^{\text{MAX}}_{i,j}$. A new parameter, $\tilde{\mu}_i$, is determined based on the amount of material that a task-unit pair can feasibly produce. Line 9 uses $\Delta\mu_i$ to achieve the minimum attainable production amount that will satisfy final demand. If μ_i already falls within the attainable range, then $\Delta\mu_i = 0$ and $\tilde{\mu}_i = \mu_i$.

After $\tilde{\mu}_i$ is calculated on for available tasks, $\mathbf{I}^{\text{A}}$, on lines 10 and 11 we update our record-keeping sets and propagate backwards again to determine ω_k for all materials that are consumed by the previously available tasks. Note that $\rho_{i,k}$ will be negative in line 12 of Algorithm 2 because materials are being consumed by tasks. Additionally, any initial inventory is considered by subtracting it from the amount of production needed. Hence, ω_k can be negative in the case of a large initial inventory. However, $v_{i,k}$ cannot be negative due the enforced lower bound of 0 on line 4. The motivation for this bound is our desire not to trivially reverse propagate negative values. Through the use of the while loop beginning on line 3, we continue propagating backwards until ω_k and $\tilde{\mu}_i$ have been calculated for all materials and tasks, respectively.

Tightening Constraints

The first tightening constraint we introduce is defined for each $i \in \mathbf{I}^{\text{P}}$ and imposes a lower bound on the sum of $X_{i,j,n}$ over all units $j \in \mathbf{J}_i$ and time periods $n \in \mathbf{N}$. A valid value for this bound is obtained by dividing $\tilde{\mu}_i$ by the maximum production rate over all $j \in \mathbf{J}_i$ and rounding up. We have

$$\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} X_{i,j,n} \geq \left\lceil \frac{\tilde{\mu}_i}{\max\limits_{j\in\mathbf{J}_i}\left\{\beta^{\text{MAX}}_{i,j}\right\}} \right\rceil \qquad \forall i \in \mathbf{I}^{\text{P}}. \quad (7)$$

We now consider materials $k \in \mathbf{K}$ for which $|\mathbf{I}^+_k| > 1$, i.e., more than one task is able to produce k. In this case, the minimum processing time for all $i \in \mathbf{I}^+_k$ can be zero since it is possible that all demand for k can be satisfied using only tasks in $\mathbf{I}^+_k \setminus \{i\}$. Hence, for each such material we compute the minimum amount of time that the tasks in $\mathbf{I}^+_k$ must collectively run in order to produce the required amount of k and use this value to bound the task execution binary variables. This gives

$$\sum_{i\in\mathbf{I}^+_k}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} X_{i,j,n} \geq \left\lceil \frac{\omega_k}{\max\limits_{i\in\mathbf{I}^+_k, j\in\mathbf{J}_i}\left\{\beta^{\text{MAX}}_{i,j}\rho_{i,k}\right\}} \right\rceil \qquad \forall k : |\mathbf{I}^+_k| > 1. \quad (8)$$

We note that when units with different production rates can process the same task, Eq. (7) may not provide a tight bound. Thus, in this case we develop an alternative to Eq. (7) by considering the production amounts of the tasks being executed rather than their production times. Specifically, for each $i \in \mathbf{I}^{\text{P}}$ we introduce a new parameter $\hat{\mu}^{\text{X}}_i$, which is similar to $\tilde{\mu}_i$ and serves as a lower bound on the sum of $\beta^{\text{MAX}}_{i,j}X_{i,j,n}$ over all units $j \in \mathbf{J}_i$ and time periods $n \in \mathbf{N}$. The resulting constraint is then:

$$\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} \beta^{\text{MAX}}_{i,j}X_{i,j,n} \geq \hat{\mu}^{\text{X}}_i \qquad \forall i \in \mathbf{I}^{\text{P}} \quad (9)$$

Recognizing that in Eq. (9) we have not considered situations in which multiple tasks can produce the same material, an additional constraint that exploits this case is written as follows:

$$\sum_{i\in\mathbf{I}^+_k}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} \rho_{i,k}\beta^{\text{MAX}}_{i,j}X_{i,j,n} \geq \omega_k \qquad \forall k : |\mathbf{I}^+_k| > 1 \quad (10)$$

We have now presented all of our proposed tightening constraints that utilize the task execution binaries, $X_{i,j,n}$, and turn our focus to constraints that incorporate the run initiation binaries, $Y^{\text{S}}_{i,j,n}$. To begin, for each task $i \in \mathbf{I}^{\text{P}}$ we can calculate the minimum number of runs of i that are needed in order to satisfy demand and use this value to bound the sum of $Y^{\text{S}}_{i,j,n}$ over all units $j \in \mathbf{J}_i$ and time periods $n \in \mathbf{N}$ as follows:

$$\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} Y^{\text{S}}_{i,j,n} \geq \left\lceil \frac{\tilde{\mu}_i}{\max\limits_{j\in\mathbf{J}_i}\left\{\beta^{\text{MAX}}_{i,j}\tau^{\text{MAX}}_{i,j}\right\}} \right\rceil \qquad \forall i \in \mathbf{I}^{\text{P}} \quad (11)$$

Situations in which multiple tasks can produce a given $k \in \mathbf{K}$ can be exploited by ensuring that, collectively, enough runs of the tasks in $\mathbf{I}^+_k$ are started in order to achieve demand. This can be enforced using the following constraint:

$$\sum_{i\in\mathbf{I}^+_k}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} Y^{\text{S}}_{i,j,n} \geq \left\lceil \frac{\omega_k}{\max\limits_{i\in\mathbf{I}^+_k\, j\in\mathbf{J}_i}\left\{\beta^{\text{MAX}}_{i,j}\rho_{i,k}\tau^{\text{MAX}}_{i,j}\right\}} \right\rceil \forall k : |\mathbf{I}^+_k| > 1 \quad (12)$$

Finally, we introduce constraints in which we consider production amounts rather than production times. These are given below:

$$\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} \beta^{\text{MAX}}_{i,j}\tau^{\text{MAX}}_{i,j}Y^{\text{S}}_{i,j,n} \geq \hat{\mu}^{\text{Y}}_i \qquad \forall i \in \mathbf{I}^{\text{P}} \quad (13)$$

$$\sum_{i\in\mathbf{I}^+_k}\sum_{j\in\mathbf{J}_i}\sum_{n\in\mathbf{N}} \rho_{i,k}\beta^{\text{MAX}}_{i,j}\tau^{\text{MAX}}_{i,j}Y^{\text{S}}_{i,j,n} \geq \omega_k \qquad \forall k : |\mathbf{I}^+_k| > 1 \quad (14)$$

Computational Performance

We conducted a test in which a total of 239 cost minimization instances were solved. We tested the original model as well as four formulations utilizing tightening constraints:

- $^{\text{n}}\text{X}$ incorporated Eqs. (7) and (8)
- $^{\text{p}}\text{X}$ incorporated Eqs. (9) and (10)
- $^{\text{n}}\text{Y}^{\text{S}}$ incorporated Eqs. (11) and (12)

- ${}^{\text{p}}\text{Y}^{\text{S}}$ incorporated Eqs. (13) and (14)

The results of our test are summarized in the performance profile displayed in Figure 3.

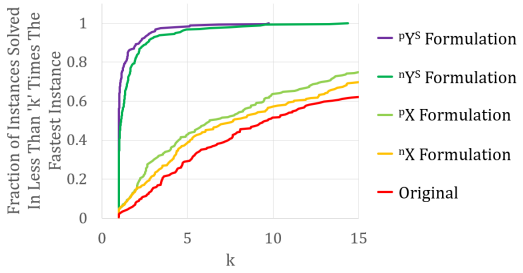

Figure 3: Performance chart comparing solve times of the original model to formulations with tightening constraints.

It is clear from Figure 3 that incorporating our proposed constraints into the model yields significant reductions in computational times. All of the models incorporating the tightening constraints significantly outperformed the original model, but the tightening constraints using the run initiation binary variables clearly performed the best. The ${}^{\text{n}}\text{Y}^{\text{S}}$ formulation solved 36.2% of the instances the fastest, and the ${}^{\text{p}}\text{Y}^{\text{S}}$ formulation solved the remaining 63.8% of instances faster than all of the other formulations. These results demonstrate the effectiveness of incorporating tightening constraints in continuous process scheduling models.

Conclusion

We have presented three novel approaches for reducing the computational time utilized when solving discrete time-based MILP models for chemical production scheduling. Two of these techniques are designed for use when modeling batch processes, and one for use when modeling continuous processes. The first of our batch process methods, a hybrid MILP/CP branch-and-cut approach, produced results comparable with current state-of-the-art approaches on about 90% of the instances we tested, but on the other 10% our approach showed significant reductions in computational effort, solving these instances in approximately a tenth of the time required by the fastest of the other approaches we compared against. Our second method for batch processes involves reformulating the base MILP model by adding so-called *record keeping variables* and constraints that place upper bounds on their values. We demonstrate that in most cases the inclusion of these variables in the MILP model results in profound speedups; often reducing the solution time to under 2% of the time used when no such variables are included in the model. Finally, the method we present for continuous processes, which involves adding constraints to the MILP model that tighten its linear programming relaxation, also reduces required computational time substantially. In fact, for approximately 50% of the instances we tested, the best performing of our proposed sets of tightening constraints results in a solution time that is less than a tenth of the time required by the original formulation.

References

Castro, P. M. and I. E. Grossmann (2014). Global optimal scheduling of crude oil blending operations with RTN continuous-time and multiparametric disaggregation. *Industrial and Engineering Chemistry Research 53*(39), 15127–15145.

Giménez, D. M., G. P. Henning, and C. T. Maravelias (2009, sep). A novel network-based continuous-time representation for process scheduling: Part I. Main concepts and mathematical formulation. *Computers and Chemical Engineering 33*(9), 1511–1528.

Harjunkoski, I. and I. E. Grossmann (2002). Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers and Chemical Engineering 26*, 1533–1552.

Maravelias, C. T. (2021). *Chemical Production Scheduling: Mixed-Integer Programming Models and Methods*. Cambridge University Press.

Méndez, C. A., G. P. Henning, and J. Cerdá (2001). An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities. *Computers and Chemical Engineering 25*, 701–711.

Merchan, A. F., H. Lee, and C. T. Maravelias (2016). Discrete-time mixed-integer programming models and solution methods for production scheduling in multistage facilities. *Computers & Chemical Engineering 94*, 387–410.

Papavasileiou, V., A. Koulouris, C. Siletti, and D. Petrides (2007). Optimize manufacturing of pharmaceutical products with process simulation and production scheduling tools. *Chemical Engineering Research and Design 85*(7 A), 1086–1097.

Sadykov, R. and L. A. Wolsey (2006). Integer programming and constraint programming in solving a multimachine assignment scheduling problem with deadlines and release dates. *INFORMS Journal on Computing 18*(2), 209–217.

Subrahmanyam, S., G. K. Kudva, M. H. Bassett, and J. F. Pekny (1996). Application of Plant Distributed Design and Computing to Batch Scheduling. *AIChE Journal 42*(6), 1648–1661.

Velez, S. and C. T. Maravelias (2013, mar). Reformulations and branching methods for mixed-integer programming chemical production scheduling models. *Industrial and Engineering Chemistry Research 52*(10), 3832–3841.

Velez, S., A. Sundaramoorthy, and C. T. Maravelias (2013, mar). Valid Inequalities Based on Demand Propagation for Chemical Production Scheduling MIP Models. *AIChE Journal 59*(3), 872–887.

Wolsey, L. A. (1997). MIP modelling of changeovers in production planning and scheduling problems. *European Journal of Operational Research 99*, 154–165.

Wu, Y. and C. T. Maravelias (2021). A general framework and optimization models for the scheduling of continuous chemical processes. *AIChE Journal 67*(10), 1–15.