On the Utility of Production Scheduling Formulations Including Record Keeping Variables

Nathan Adelgren[a,*], Christos T. Maravelias[a,b]

[a] *Andlinger Center for Energy and the Environment, Princeton University, Princeton, 08544, NJ, United States*
[b] *Department of Chemical and Biological Engineering, Princeton University, Princeton, 08544, NJ, United States*

Abstract

In this work we propose several reformulations of a general discrete-time mixed-integer formulation for production scheduling problems. Each reformulation involves the addition of new integer variables that we refer to as record keeping variables. Computational results show that the required CPU time is greatly reduced when record keeping variables are present in the optimization model. Additionally, we assess which of the individual aspects of traditional branch-and-bound solution procedures benefit from the inclusion of record keeping variables.

Keywords: production scheduling, mixed integer programming, reformulation, record keeping variables

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT Authorship Contribution Statement

Nathan Adelgren: Conceptualization, Methodology, Software, Validation, Formal Analysis, Visualization, Writing - Original Draft, Writing - Review & Editing. **Christos T. Maravelias**: Conceptualization, Supervision, Funding Acquisition, Writing - Review & Editing.

*Corresponding author
Email addresses: na4592@princeton.edu (Nathan Adelgren), maravelias@princeton.edu (Christos T. Maravelias)

1. Introduction

In recent years considerable attention has been given to the utilization of mathematical optimization to aid decision makers in developing high quality production schedules. Employing such techniques not only improves the quality of the schedules used in practice, but also allows for better asset utilization as they reduce the amount of human-interaction required when designing a production schedule. Production scheduling is a necessary component of decision making in almost every manufacturing-related industry and, as such, much work has been done in order to develop specialized modeling techniques and optimization methods that account for the various process characteristics present in each application within each industry. We point the interested reader to the works of Harjunkoski et al. [1] and Georgiadis et al. [2] for extensive lists of the many works dedicated to applications of production scheduling within various industries.

As one might imagine, though, the development of sophisticated optimization models capable of aiding one in making complicated decisions often comes with a price. Namely, these optimization models can become extremely large and complex, to the point where they may become intractable. Examples of complex process characteristics for which optimization models and techniques have recently been proposed include required changeover times between tasks (Avalos-Rosales et al. [3], Velez et al. [4], Cafaro and Grossmann [5], Kramer et al. [6], Berkhout et al. [7], Castro [8]), task interruption at specified break points (Castro et al. [9]), and tracking and limiting utility consumption (Wang et al. [10]). In each of these cited works, considerable effort is given to developing modeling or optimization techniques that combat the added computational effort required in order to solve the scheduling problem once the associated process characteristics are accounted for. We note that large CPU time requirements are particularly troublesome in cases in which a production scheduling problem needs to be iteratively resolved in order to obtain updated schedules [11, 12, 13].

Traditionally, mixed-integer linear programming (MILP) methods have been the primary tool used in the literature for modeling and solving production scheduling problems. Most of the MILP models that have been employed can be classified as either continuous-time-based models or discrete-time-based models. As these names suggest, in the former class of problems time is represented using a continuous scale, whereas in the latter class time is represented using a discrete set of time points that are typically evenly spaced. Floudas and Lin [14] and Sundaramoorthy and Maravelias [15] provide excellent overviews of these two types of models as well as thorough comparisons of their strengths and weaknesses. The primary strength of continuous formulations is that they are relatively small and produce extremely accurate solutions. It is often difficult to expand them in order to account for complex process characteristics, though, and this is their main shortcoming. On the other hand, it is often straightforward to

adjust discrete formulations in order to account for complicated process characteristics. However, they often require relatively fine discretizations of time in order to obtain accurate solutions, and this can cause models to become quite large and utilize significant CPU time.

Several works in the literature have been devoted to the study and development of alternatives and/or extensions to MILPs that can be used to model and solve challenging scheduling problems. Examples of approaches that have been proposed include column generation (Chen and Powell [16, 17], Gélinas and Soumis [18], Lopes and de Carvalho [19], Van den Akker et al. [20], Ghoniem and Farhadi [21], Xiong et al. [22]), constraint programming (Jain and Grossmann [23], Baptiste et al. [24], Bockmayr and Pisaruk [25], El Khayat et al. [26], Sadykov and Wolsey [27], Zeballos et al. [28], Edis and Ozkarahan [29], Ham [30], Gedik et al. [31], Meng et al. [32], Awad et al. [33]), and disjunctive programming (Castro and Grossmann [34], Castro et al. [35], Castro and Marques [36], Mostafaei et al. [37], Wu et al. [38]).

We note, however, that most of the sophisticated techniques mentioned above are specifically designed for use with a single application. For this reason, we focus on a general, widely applicable class of scheduling problems that we model as a discrete-time-based MILP. We choose to employ a discrete-time model in order to allow for simple extensions of our proposed methodology that account for the incorporation of complex process characteristics into the MILP model we employ. Specifically, we focus this work on MILP modeling techniques that allow one to significantly reduce the computational overhead of solving general production scheduling problems. Our motivation is that, by improving the quality of the formulation used to represent the general production scheduling problem that serves as the foundation for more complex variants, we may achieve improved performance for a wide variety of classes of challenging problems.

Our current work is highly motivated by that of Velez and Maravelias [39] in which the authors showed that the incorporation of a relatively small number of additional integer variables and associated constraints into the formulation can significantly reduce the CPU time required to solve instances of that problem. We extend this work by considering additional integer variables and associated constraints that can be incorporated alongside those proposed in [39] in order to achieve further reductions in utilized computational resources.

The remainder of this article is organized as follows. In Section 2 we present the notation that we employ throughout the work as well as a commonly used MILP formulation. In Section 3 we review the primary reformulation of the aforementioned MILP model that is proposed in [39] and also propose several additional reformulations. We then assess the strengths of the proposed reformulations and provide a comparison with the primary reformulation of [39] in Section 4. Section 5 contains a study of the impact of imposing a hierarchical set of branching priorities on the various binary and integer variables. A de-

tailed study of the effects of our proposed reformulations on several phases of a traditional branch-and-bound (BB) solution procedure is presented in Section 6. Section 7 contains the results of a set of tests designed to assess the utility of our proposed reformulations when applied to complex problem variations. Finally, we provide concluding remarks in Section 8.

2. Background

We consider a general manufacturing facility and employ a state-task network (STN) for its representation (see, for example, [40, 41]). The rest of this section is divided into two parts. In the first we present the notation that we employ, and in the second we provide a MILP formulation for the production scheduling problem that serves as the foundation for the alternate formulations we propose in Section 3.

2.1. Notation

We employ the following convention for notation:

Sets	bold, upper-case, roman letters
Indices	lower-case roman letters
Parameters	lower-case greek letters
Variables	non-bold, upper-case, roman letters

The STN representation of a facility then relies on the elements (and their associated sets)

$i \in \mathbf{I}$	tasks,
$j \in \mathbf{J}$	units, and
$k \in \mathbf{K}$	materials;

the sets

$\mathbf{I}_k^+ / \mathbf{I}_k^-$	tasks producing/consuming material k and
$\mathbf{J}_i$	units permitted to carry out task i;

and several related parameters, defined in Section 2.2.

Specifically, the STN is the directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where the set $\mathbf{V}$ of vertices is given by $\mathbf{V} := \mathbf{K} \cup \mathbf{I}$ and for each pair (i, k) with $i \in \mathbf{I}$ and $k \in \mathbf{K}$, there exists an arc $e_{ik} \in \mathbf{E}$ from i to k if and only $i \in \mathbf{I}_k^+$ and an arc $e_{ki} \in \mathbf{E}$ from k to i if and only $i \in \mathbf{I}_k^-$. We note that processing units (and other shared utilities, when applicable) are represented implicitly in the STN through: (i) the mappings implied by subset definitions, and (ii) the values of corresponding parameters. An example STN representing a facility that creates two products (k_8 and k_9) from three feedstocks (k_1, k_2 and k_3), taken from [40], is displayed in Figure 1.

2.2. Problem Formulation

We now present a MILP model for the scheduling problem, as first proposed by Shah et al. [41]. To begin, we discretize the scheduling horizon η by dividing it into n periods of length $\delta = \eta/n$. We then define the set $\mathbf{T} = \{0, 1, \ldots, n\}$ of time points, where each $t \in \mathbf{T}$ corresponds to the point in time that is $t\delta$ time units beyond the start

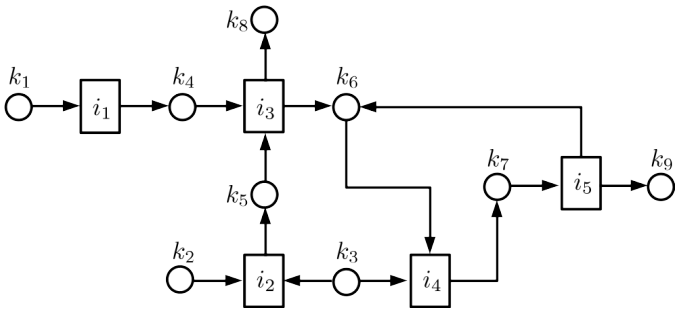


Figure 1: Example STN – Kondili et al. [40]

Task-unit mapping: $\mathbf{J}_{i_1} = \{j_1\}$; $\mathbf{J}_{i_2} = \mathbf{J}_{i_3} = \mathbf{J}_{i_4} = \{j_2, j_3\}$; $\mathbf{J}_{i_5} = \{j_4\}$.

of the horizon. We note that each $t \in \mathbf{T} \setminus \{0\}$ can also be used to identify the time *period* $[(t-1)\delta, t\delta)$. For use in the MILP model, we define the parameters

$\beta_j^{\min}/\beta_j^{\max}$	minimum/maximum capacity of unit j,
ξ_{kt}	net shipment of material k at time t (positive for incoming quantities, negative for outgoing),
ρ_{ik}	conversion coefficient of material k produced or consumed by task i (positive for production, negative for consumption),
τ_{ij}	time required to process task i in unit j, and
$\chi_k^{\max}$	maximum amount of material k that can be stored;

and the variables

$X_{ijt} \in \{0,1\}$	1 if task i begins in unit j at time t,
$B_{ijt} \in \mathbb{R}_+$	batch size of task i processed in unit j at time t, and
$I_{kt} \in \mathbb{R}_+$	inventory level of material k during time period t.

A general version of the MILP model is then given by

$$\mathcal{P} := \min \quad f(\cdot)$$

$$\text{s.t.} \quad \sum_{i:\, j \in \mathbf{J}_i} \sum_{t'=t-\lceil \tau_{ij}/\delta \rceil + 1}^{t} X_{ijt'} \leq 1 \qquad \forall\, j \in \mathbf{J}, t \in \mathbf{T} \quad (1)$$

$$\beta_j^{\min} X_{ijt} \leq B_{ijt} \leq \beta_j^{\max} X_{ijt} \qquad \forall\, i \in \mathbf{I}, j \in \mathbf{J}_i, t \in \mathbf{T} \quad (2)$$

$$I_{k(t+1)} = I_{kt} + \sum_{i \in \mathbf{I}_k^+} \sum_{j \in \mathbf{J}_i} \rho_{ik} B_{ij(t-\lceil \tau_{ij}/\delta \rceil)} + \sum_{i \in \mathbf{I}_k^-} \sum_{j \in \mathbf{J}_i} \rho_{ik} B_{ijt} + \xi_{kt} \leq \chi_k^{\max} \qquad \forall\, k \in \mathbf{K}, t \in \mathbf{T}. \quad (3)$$

We note that in problem $\mathcal{P}$: (i) $f(\cdot)$ denotes an arbitrary function that we will specify later; (ii) Equation (1) ensures that at most one task is performed in each unit at each time point, and moreover, that once a task begins in a unit, that unit can perform no other task until the first has been completed; (iii) Equation (2) ensures that any batch of a task performed within a unit remains within that unit's minimum and maximum capacity; and (iv) Equation (3) serves to keep track of the quantity of each material that is in inventory and ensures that the amount of any given material in inventory does not exceed the maximum allowable storage for that material. We also point out that

in a slight abuse of notation, Equation (3) defines variables I_{k0} and $I_{k(n+1)}$ for each $k \in \mathbf{K}$ even though we have not formally introduced the concepts of 0^{th} and $(n+1)^{\text{th}}$ time periods. As the former would be prior to the start of the scheduling horizon, for each $k \in \mathbf{K}$ we treat I_{k0} as a parameter representing initial inventory. Similarly, as the latter time period would be beyond the end of the scheduling horizon, we take $I_{k(n+1)}$ to represent the final inventory of each material $k \in \mathbf{K}$.

Another point worth mentioning is that the task time τ_{ij} for a given $i \in \mathbf{I}$ and $j \in \mathbf{J}_i$ may not be an integer multiple of δ. In this case, a certain amount of *discretization error* is introduced into problem $\mathcal{P}$ in order to ensure that no unit processes more than one task at a time and that material balances are handled appropriately. This can be seen by the use of $\lceil \tau_{ij}/\delta \rceil$ in Equations (1) and (3). Practically, the implication is that each execution of task i in unit j will be followed by $\delta\lceil \tau_{ij}/\delta \rceil - \tau_{ij}$ units of idle time.

For the majority of this work we consider two variants of problem $\mathcal{P}$: (i) cost minimization, and (ii) profit maximization. We note that makespan minimization was also considered, but we do not discuss it further here due to the relatively small CPU times observed when solving these problems. The interested reader can find results for makespan minimization in the Supplementary Material. For cost minimization, we introduce a parameter γ_{ij} to represent the cost of performing task i in unit j and set $f(\cdot) = \sum_{i \in \mathbf{I}} \sum_{j \in \mathbf{J}_i} \gamma_{ij} X_{ijt}$. For profit maximization, we again utilize γ_{ij}, introduce an additional parameter π_k representing the revenue obtained from selling one unit of material k, and set $f(\cdot) = \sum_{i \in \mathbf{I}} \sum_{j \in \mathbf{J}_i} \gamma_{ij} X_{ijt} - \sum_{k \in \mathbf{K}} \pi_k I_{k(n+1)}$. Recognize that, although problem $\mathcal{P}$ is formulated as a minimization problem, we have defined $f(\cdot)$ here as cost minus revenue, and minimizing this quantity is equivalent to maximizing revenue minus cost.

3. Reformulations

We divide this section into two subsections. In Section 3.1 we discuss a reformulation from the literature that serves as a foundation for our current work. Our proposed extensions are then presented in Section 3.2.

3.1. Previous Work

In [39], the authors propose reformulating problem $\mathcal{P}$ by adding an integer variable N_{ij} that represents the number of times task i is carried out in unit j. Specifically, problem $\mathcal{P}$ is modified by adding the constraint

$$\sum_{t \in \mathbf{T}} X_{ijt} = N_{ij} \qquad \forall\, i \in \mathbf{I}, j \in \mathbf{J}_i \tag{4}$$

and bounding N_{ij} as

$$0 \leq N_{ij} \leq \lfloor \eta/\tau_{ij} \rfloor \qquad \forall\, i \in \mathbf{I}, j \in \mathbf{J}_i. \tag{5}$$

The authors of [39] state that their motivation for proposing this reformulation is the fact that for many instances of problem $\mathcal{P}$, there are multiple feasible solutions having the same objective value. Specifically, they argue that many of these solutions having the same objective value share the same (task, unit) pair assignments, but that these assignments occur at different time points in different solutions.

Hence, they suspect (and later confirm) that branching on N_{ij} can lead to faster solution times due to the fact that each such branching decision may eliminate multiple sub-optimal solutions simultaneously. As a result, tightening of the dual bound generally occurs after the exploration of fewer nodes when branching is permitted to occur on N_{ij} as opposed to only X_{ijt}.

We conjecture, however, that the benefits of defining N_{ij} and adding Equations (4) and (5) to $\mathcal{P}$ extend beyond those mentioned by Velez and Maravelias [39]. Specifically, we believe that it is not only the branching phase of BB that benefits from these additions. This line of reasoning is explored further in Section 6 and serves as the motivation for the extended reformulations we propose in the following subsection.

3.2. Further Considerations

In this subsection we define several new integer variables that were not considered in [39], and present associated constraints and bounds that, when added to the structure of $\mathcal{P}$, may serve to further enhance the performance of BB beyond the improvements observed by Velez and Maravelias [39]. 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,

$N_t \in \mathbb{Z}^+$ the number of tasks performed at time t, and

$N \in \mathbb{Z}^+$ the total number of tasks performed,

that can be incorporated into $\mathcal{P}$ using the following constraints and bounds:

$$N_i = \sum_{j \in \mathbf{J}_i} \sum_{t \in \mathbf{T}} X_{ijt} \qquad \forall\, i \in \mathbf{I} \tag{6}$$

$$0 \leq N_i \leq \sum_{j \in \mathbf{J}_i} \left\lfloor \frac{\eta}{\tau_{ij}} \right\rfloor \qquad \forall\, i \in \mathbf{I} \tag{7}$$

$$N_j = \sum_{i:\, j \in \mathbf{J}_i} \sum_{t \in \mathbf{T}} X_{ijt} \qquad \forall\, j \in \mathbf{J} \tag{8}$$

$$0 \leq N_j \leq \left\lfloor \frac{\eta}{\min\limits_{i:\, j \in \mathbf{J}_i} \{\tau_{ij}\}} \right\rfloor \qquad \forall\, j \in \mathbf{J} \tag{9}$$

$$N_t = \sum_{i \in \mathbf{I}} \sum_{j \in \mathbf{J}_i} X_{ijt} \qquad \forall\, t \in \mathbf{T} \tag{10}$$

$$0 \leq N_t \leq \min\{|\mathbf{I}|, |\mathbf{J}|\} \qquad \forall\, t \in \mathbf{T} \tag{11}$$

$$N = \sum_{i \in \mathbf{I}} \sum_{j \in \mathbf{J}_i} \sum_{t \in \mathbf{T}} X_{ijt} \tag{12}$$

$$0 \leq N \leq \min \left\{ \sum_{i \in \mathbf{I}} \sum_{j \in \mathbf{J}_i} \left\lfloor \frac{\eta}{\tau_{ij}} \right\rfloor, \sum_{j \in \mathbf{J}} \left\lfloor \frac{\eta}{\min\limits_{i:\, j \in \mathbf{J}_i} \{\tau_{ij}\}} \right\rfloor \right\} \tag{13}$$

The variables N_i, N_j, N_t, and N can be incorporated into $\mathcal{P}$ simultaneously using Equations (6)–(13), or individually by including Equations (6)–(7) for N_i, Equations (8)–(9) for N_j, Equations (10)–(11) for N_t, and Equations (12)–(13) for N. Moreover, as any subset of $\{N_{ij}, N_i, N_j, N_t, N\}$ can be incorporated into $\mathcal{P}$, we have $2^5 - 1 = 31$ reformulations of $\mathcal{P}$ to consider. Recognize that each of the variables N_{ij}, N_i, N_j, N_t, and N serves to *keep record* of a quantity of interest when solving $\mathcal{P}$. As such, throughout the

remainder of this work we refer to each of these variables as a *record keeping variable*. In Section 4 we present the results of a study in which we compare the performance of BB when applied to $\mathcal{P}$ and a carefully chosen subset of the proposed reformulations that involve these record keeping variables.

4. Initial Assessment of Reformulations

4.1. Preliminaries

In this section we present the results of two computational studies. The tests conducted throughout the rest of this work are performed using three sets of instances of the scheduling problem. The first two sets are obtained from `minlp.org` and from the authors of [39], respectively. The final set was randomly generated. We note that, while there is only one base instance obtained from `minlp.org`, the problem structure of this instance is designed in such a way that modifying the event horizon η results in a distinct variant of the instance. Thus, we utilize seven variants of this instance by setting $\eta = 24, 28, 32, 36, 40, 44,$ and 48. The authors of [39] also graciously shared with us eight instances of the scheduling problem, derived from seven unique network structures that were originally presented by Kondili et al. [40], Papageorgiou and Pantelides [42], Maravelias [43], and Velez et al. [44]. In order to expand our test set, we randomly generated 100 additional instances. We make these instances available for the interested reader at `https://github.com/Nadelgren/RNBBS_Instances`.

All of the computational tests described in this work, with the exception of Section 6.4, were conducted using Princeton University's Della cluster (`https://researchcomputing.princeton.edu/systems/della`). Specifically, tests were conducted using nodes on a Dell Linux cluster running Springdale Linux 8. Each node had 2.4 – 2.8 GHz and 128 – 768 GB of RAM. Each job submitted to the cluster utilized 4 threads, with a request for 16 GB of RAM per thread. All tests for individual instances were run using identical hardware in order to ensure a reliable comparison between the various reformulations. All instances were solved using CPLEX 20.1 [45] via GAMS 36.1.0 and an execution time limit of 5 hours was imposed for each instance. We note that in a small number of cases, the processing of an instance on the cluster was killed prior to reaching the 5 hour time limit because memory utilization became too large. For reporting purposes, we record such cases as having failed to solve in 5 hours and assign an optimality gap of 100%.

We adopt a notation that allows us to simultaneously specify the CPLEX settings and (re)formulation of $\mathcal{P}$ used for each test run. Specifically, we utilize a string comprised of two substrings, separated by a period. The first substring consists of a single character specifying the CPLEX settings used, and the second substring consists of a set of characters indicating which, if any, record keeping variables are added to $\mathcal{P}$. When specifying CPLEX settings, we utilize one of the following characters:

N	CPLEX default settings were used
P	priority settings were employed for selecting the branching variable

When specifying a formulation, we utilize the following characters:

N	no additional variables/constraints are added to $\mathcal{P}$
B	N_{ij} and Equations (4)–(5) are added to $\mathcal{P}$
I	N_i and Equations (6)–(7) are added to $\mathcal{P}$
J	N_j and Equations (8)–(9) are added to $\mathcal{P}$
T	N_t and Equations (10)–(11) are added to $\mathcal{P}$
A	N and Equations (12)–(13) are added to $\mathcal{P}$

Hence, the string "N.BIJ", for example, indicates that default CPLEX settings were used and that N_{ij}, N_i, and N_j were added to $\mathcal{P}$ using Equations (4)–(9).

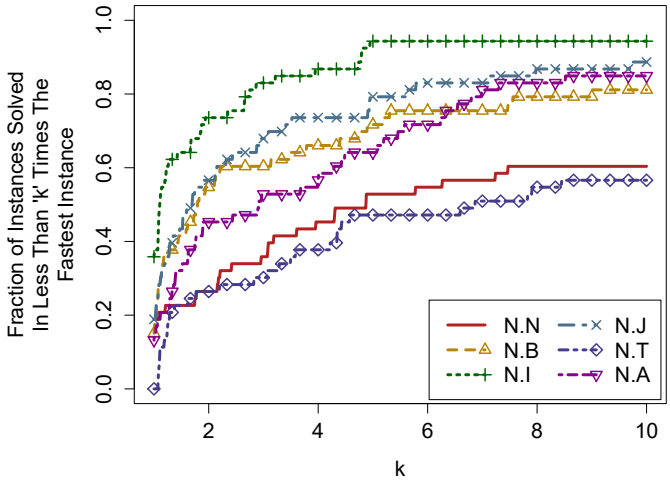
In the remainder of this section we present the results of a computational test designed to determine which, if any, of the record keeping variables should be added to the structure of $\mathcal{P}$. For the sake of space, we present two carefully selected subsets of the results for this test. The first is presented in Section 4.2 and the second in Section 4.3. Each subset of results is presented in two parts: (i) a figure displaying performance profiles of relative CPU time for instances that all considered formulations were able to solve in under 5 hours, and (ii) a table displaying CPU time and optimality gap results for instances that at least one of the considered formulations was able to solve in under 5 hours and at least one was unable to solve in under 5 hours.

4.2. Reformulations Containing at Most One Record Keeping Variable

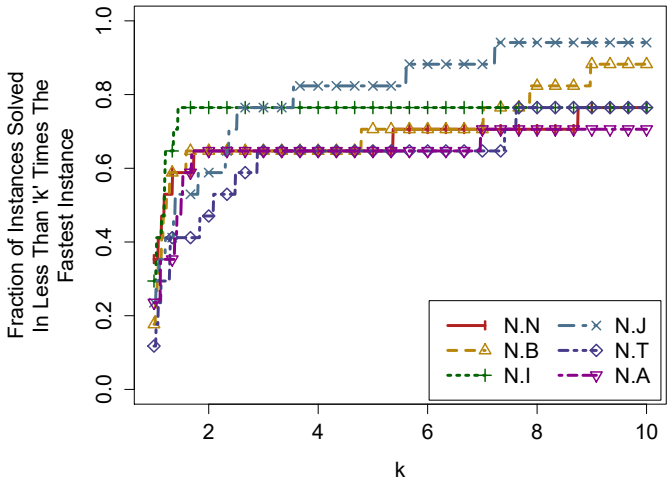
Here we present results for N.N, N.B, N.I, N.J, N.T, and N.A. Figure 2 contains a performance profile of relative CPU time for instances that all formulations were able to solve in under 5 hours and Table 1 displays CPU time and optimality gap results for instances that at least one of the formulations was able to solve in under 5 hours and at least one was not.

For the results displayed in Figure 2, we see that N.I displayed the strongest performance for cost minimization and both N.I and N.J displayed strong performance for profit maximization. We also note that N.B performed reasonably well for both objective types, while N.T and N.N performed relatively poorly for both objective types. Interestingly, the performance of N.A was reasonable for cost minimization, but poor for profit maximization. We do remind the reader, however, that the results displayed in Figure 2 should be interpreted with some caution as the results displayed therein are for instances that all of the considered formulations were able to solve in under five hours. Hence, these instances are, in some sense, the easiest of the instances considered.

We now turn our attention to Table 1 which contains results for instances that at least one formulation was able to solve in under five hours and at least one was not. As such, we view these instances as being of medium difficulty.



(a) Cost



(b) Profit

Figure 2: Performance profiles of CPU time (s) for formulations containing at most one additional variable – Instances considered are those for which all formulations solved the instance in 5hrs

Table 1: Results for instances for which at least one formulation failed to solve the instance in 5hrs and at least one formulation successfully solved the instance in 5hrs

	Fraction Solved	Avg Rel Time		Avg Rel Gap	
		Cost			
N.N	15/39	90.7	(235.9)	6.1	(16.0)
N.B	20/39	64.1	(124.9)	6.2	(12.1)
N.I	34/39	3.6	(4.1)	1.4	(1.6)
N.J	25/39	7.4	(11.6)	7.3	(11.4)
N.T	4/39	65.3	(636.6)	20.8	(203.1)
N.A	14/39	121.7	(339.0)	26.0	(72.4)
		Profit			
N.N	8/41	413.6	(2119.6)	58.2	(298.4)
N.B	19/41	148.0	(319.4)	158.8	(342.6)
N.I	25/41	11.6	(19.0)	83.4	(136.7)
N.J	23/41	18.9	(33.7)	6.3	(11.3)
N.T	10/41	323.3	(1325.5)	229.0	(938.9)
N.A	16/41	200.9	(514.8)	408.9	(1047.9)

(Gray): (Average Relative Value) ÷ (Fraction Solved)

We note that because most of the considered instances come from different network structures and because the time required to solve each instance can vary so drastically, we do not consider actual CPU times or optimality gaps, but rather relative versions of each, and we then report averages for these two relative values. Specifically, we

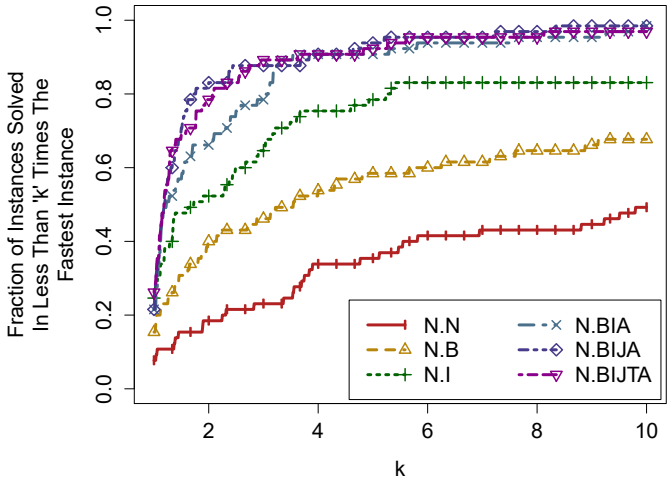
compute relative time as

$$\text{Relative Time For Instance } \ell = \frac{\text{CPU Time For Instance } \ell}{\text{Minimum of CPU Times For All Instances}}$$

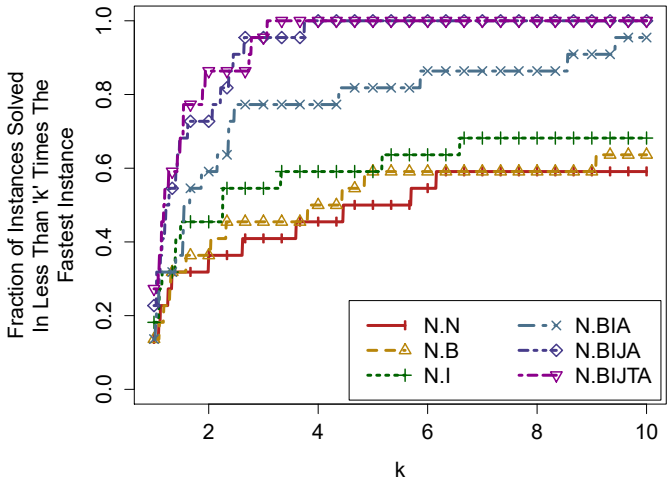
and relative optimality gap in an analogous way. We point out that reporting results in this way is consistent with the way results are displayed in performance profiles (see [46] for more information). Recognize that relative values computed as described above will always be greater than or equal to one, with values close to one indicating relatively good performance and values much larger than one indicating relatively poor performance. We also make the following important notes about the results displayed in Table 1: (i) relative CPU times are only averaged over instances that were solved in under five hours, and (ii) relative optimality gaps are only averaged over instances that *failed* to solve in under five hours. Together, points (i) and (ii) indicate that while relative CPU time and relative optimality gap may be better metrics than their absolute counterparts, both are still flawed in the sense that neither takes into consideration the number of instances that were actually successfully solved in under five hours. In an attempt to take the number of successfully solved instances into consideration, we divide each relative value by the fraction of instances solved and report this value in gray to the right of the relative value. As an example of the usefulness of this proposed metric, consider the N.T row of the cost minimization section of Table 1. Note that when considering relative CPU time, one may conclude that N.T performs relatively well for these types of instances if they overlook the fact that N.T only solved four of the considered 39 instances in under five hours. Using the proposed metric, however, the modified relative time for N.T is significantly higher, greatly exceeding the values associated with all other formulations. Hence, this provides some indication that the performance of N.T should be considered poor even though its reported average relative time is not extremely large.

4.3. Reformulations Containing More Than One Record Keeping Variable

We now shift our focus to a comparison of N.N, N.B, N.I, N.BIA, N.BIJA, and N.BIJTA. We note that N.N and N.B are included here because N.N provides a comparison with the original formulation of $\mathcal{P}$ and N.B provides a comparison with the primary formulation considered in [39]. N.I is included because, among the reformulations considered in Section 4.2, it is the reformulation that showed the most promising results for cost minimization instances and it is one of the top performing reformulations for profit maximization instances. We include only formulations N.BIA, N.BIJA, and N.BIJTA rather than all formulations containing two or more record keeping variables in order to save space and because of the strong performance that N.BIA, N.BIJA, and N.BIJTA displayed relative to that of the other formulations containing two or more record keeping variables. Results for the aforementioned formulations are presented in Figure 3 and Table



(a) Cost



(b) Profit

Figure 3: Performance profiles of CPU time (s) for formulations N.N, N.B, N.I, N.BIA, N.BIJA, and N.BIJTA – Instances considered are those for which all formulations solved the instance in 5hrs

2. The information presented in each is analogous to that contained in Figure 2 and Table 1, respectively. From the results displayed in Figure 3 and Table 2, we observe: (i) the performance of both N.N and N.B is relatively poor in all cases, (ii) N.I performs moderately well, and (iii) all of N.BIA, N.BIJA, and N.BIJTA perform quite well on all instances, though N.BIA consistently displays the weakest performance of the three. Additionally, it is worth noting

Table 2: Results for instances for which at least one formulation failed to solve the instance in 5hrs and at least one formulation successfully solved the instance in 5hrs

	Fraction Solved	Avg Rel Time		Avg Rel Gap	
		Cost			
N.N	5/39	246.0	(1918.5)	5.4	(42.1)
N.B	8/39	188.6	(919.3)	6.5	(31.8)
N.I	22/39	25.7	(45.5)	2.2	(3.8)
N.BIA	30/39	29.9	(38.8)	1.9	(2.5)
N.BIJA	35/39	2.0	(2.2)	2.6	(2.9)
N.BIJTA	34/39	14.7	(16.8)	2.8	(3.3)
		Profit			
N.N	3/64	831.6	(17740.2)	44.7	(953.5)
N.B	13/64	47.6	(234.4)	304.9	(1501.0)
N.I	20/64	18.9	(60.4)	321.1	(1027.5)
N.BIA	42/64	107.3	(163.5)	468.2	(713.5)
N.BIJA	55/64	3.5	(4.0)	44.1	(51.4)
N.BIJTA	52/64	6.0	(7.4)	7.5	(9.2)

(Gray): (Average Relative Value) ÷ (Fraction Solved)

that N.BIJA solves more instances in under 5 hours than any of the other formulations.

The results presented in this section show that reformulations of $\mathcal{P}$ containing a single record keeping variable can be useful in reducing the CPU time required to compute an optimal solution to $\mathcal{P}$. While this result is to be expected based on the previous work of Velez and Maravelias [39], we note that the reformulations considered herein (namely, N.I and N.J) offer significant reductions in CPU time beyond those observed in Velez and Maravelias [39]. Moreover, our results further show that additional reductions in CPU time can often be obtained by using extended reformulations of $\mathcal{P}$ in which combinations of record keeping variables are simultaneously added to $\mathcal{P}$. We point out that, while it may be difficult to know a priori which formulation of $\mathcal{P}$ will produce the best results for a given instance, in many real-world applications the scheduling problem $\mathcal{P}$ is solved periodically, rather than only once. In such cases the underlying network structure defining $\mathcal{P}$ often remains unchanged and the modifications to $\mathcal{P}$ result from changes in parameter values. Hence, it is quite likely that, in practice, once a decision maker determines a formulation of $\mathcal{P}$ that offers strong performance for their specific application, the same formulation will continue to provide strong results even as parameter values are adjusted.

5. Prioritized Branching

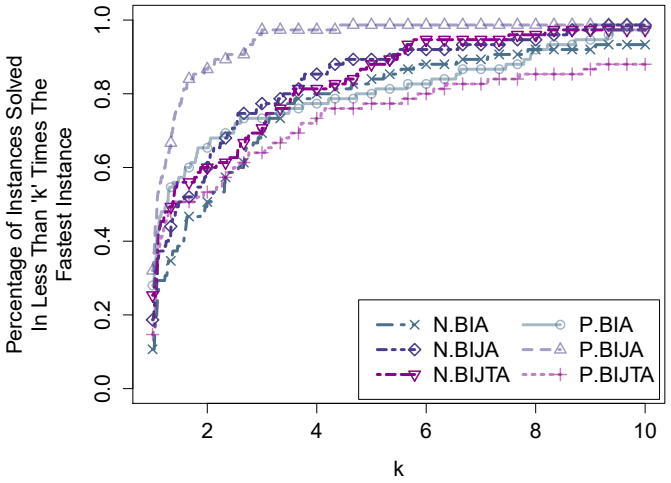
We now consider the impact of imposing a hierarchical set of branching priorities on the various binary and integer variables present in $\mathcal{P}$. Specifically, we adjust the default parameter settings in GAMS so that each type of binary and/or integer variable is given a priority score. In this way, each time a branching decision is made we ensure that the branching variable is selected from the set of variables having the highest priority score among all variables that are candidates for branching.

Before presenting any results, we make two notes. First, we do not consider the case in which the variable X_{ijt} is given a higher priority score than any of N_{ij}, N_i, N_j, N_t or N. To see why, consider the variable N_i, for example, and suppose that the integrality restrictions on N_i and X_{ijk} are relaxed. Observe from Equation (6) that if there exists an $i' \in \mathbf{I}$ such that $N_{i'}$ is fractional, then there must exist at least one $j' \in \mathbf{J}$ and $t' \in \mathbf{T}$ such that $X_{i'j't'}$ is fractional. Hence, if X_{ijt} is given a higher priority score than any of N_{ij}, N_i, N_j, N_t or N, then branching will never occur on the record keeping variable(s) having a lower priority score than X_{ijt} and it would therefore be pointless to include these variables in $\mathcal{P}$. Second, all record keeping variables are given a priority score that is exactly one greater than the priority score of X_{ijt}. We did run preliminary tests in which each of the variables X_{ijt}, N_{ij}, N_i, N_j, N_t and N were assigned unique priority scores, but the results were not significantly different from those we

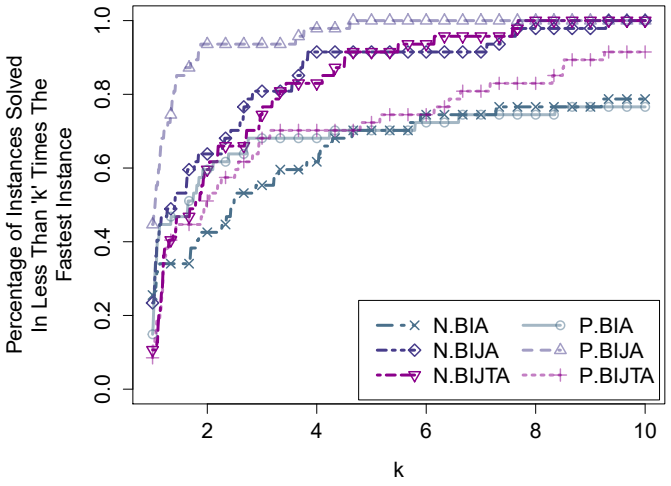
present here.

The formulations we now consider are N.BIA, N.BIJA, N.BIJTA and their respective prioritized counterparts P.BIA, P.BIJA, and P.BIJTA. For the sake of space, and because of the relatively poor performance of N.N, N.B, and N.I observed in Section 4.3, we no longer consider these formulations. Results are presented in Figure 4 and Table 3. We note that to aid in distinguishing between formulations containing prioritized variables and those that do not, in Figure 4 each formulation containing prioritized variables is presented using the same color as its non-prioritized counterpart, but with an opacity setting of 0.5.

From Figure 4 and Table 3 we observe that P.BIJA clearly outperforms all other approaches for instances that all approaches are able to solve in under 5 hours. Interestingly, though, for instances that at least one approach failed to solve and one approach successfully solved in under 5 hours, both N.BIJA and N.BIJTA perform comparably to, or perhaps even better than, P.BIJA.



(a) Cost



(b) Profit

Figure 4: Performance profiles of CPU time (s) for formulations N.BIA, N.BIJA, N.BIJTA, P.BIA, P.BIJA, and P.BIJTA – Instances considered are those for which all formulations solved the instance in 5hrs

6. BB Impacts

We now shift our focus from the overall solution time for each formulations and instead consider the impact of each formulation on each of the following aspects of BB: (i) preprocessing, (ii) heuristics, and (iii) branching. Our goal here is to determine *why* the reductions in CPU time observed in Section 4 are occurring, or said differently, which specific aspects of BB are able to exploit the presence of the record keeping variables in order to reduce the overall solu-

Table 3: Results for instances for which at least one formulation failed to solve the instance in 5hrs and at least one formulation successfully solved the instance in 5hrs

	Fraction Solved	Avg Rel Time		Avg Rel Gap	
		Cost			
N.BIA	20/31	60.9	(94.3)	2.4	(3.7)
N.BIJA	26/31	4.3	(5.1)	1.2	(1.5)
N.BIJTA	24/31	29.2	(37.7)	2.0	(2.5)
P.BIA	16/31	22.3	(43.3)	1.8	(3.4)
P.BIJA	23/31	3.5	(4.7)	1.6	(2.2)
P.BIJTA	8/31	11.0	(42.6)	2.9	(11.1)
		Profit			
N.BIA	16/39	17.1	(41.7)	509.0	(1240.8)
N.BIJA	31/39	9.8	(12.3)	418.3	(526.3)
N.BIJTA	27/39	11.0	(15.8)	10.7	(15.5)
P.BIA	21/39	132.3	(245.8)	563.7	(1046.8)
P.BIJA	29/39	8.0	(10.7)	2.5	(3.4)
P.BIJTA	5/39	2.0	(15.9)	31.7	(247.4)

(Gray): (Average Relative Value) ÷ (Fraction Solved)

tion time. We divide the remainder of this section into four subsections, one for each of the aforementioned BB components, and one in which we provide a deeper analysis of our obtained results for a small subset of the considered instances and also compare the performance of CPLEX to that of other state-of-the-art MILP solvers. For all tests discussed in the following dialogue, we compare formulations N.N, N.BIA, N.BIJA, and N.BIJTA. We include N.N as it serves as a base case, showing the performance when no record keeping variables are added to $\mathcal{P}$, and we include N.BIA, N.BIJA, and N.BIJTA due to the relatively strong performance they displayed in Sections 4 and 5. We also note that we did conduct two preliminary tests in which we sought to determine whether or not the proposed formulations had any impact on the initial LP bound or on the presolve phase of BB. To test the former, we turned off the presolve functions of CPLEX, solved the LP relaxation of each formulation, and compared the resulting objective values. In all cases, all formulations resulted in the same objective value, showing that the initial LP bound is unchanged for any of the reformulations, i.e, none of the reformulations are tighter than the original. To test the latter, we turned the presolve functions of CPLEX back on, solved the LP relaxation of the presolved version of each formulation, and compared the resulting objective values. In all cases, all formulations resulted in the same presolved LP bound. We do note, however, that for approximately 12% of instances the presolved LP bound improved over its non-presolved counterpart. Hence, we conclude that

all of the formulations contain identical information that can be exploited during the presolve phase of BB in order to obtain a tighter LP bound. Please note that we do not claim that there are no differences in the way that presolve functions for each of the formulations. Certainly presolve does function differently as each formulation contains different sets of variables and, as a result, row, column, and nonzero reductions can be significantly different among the formulations. We claim only that these reductions, though different, did not result in differences in initial LP bounds for any of the instances we studied.

6.1. Preprocessing

We note here that by *preprocessing* we mean all operations that are performed by a solver after the completion of presolve, but prior to the start of branching, that serve to further tighten the dual bound. To our knowledge, in CPLEX, preprocessing consists of probing, bound tightening, cut generation, and possibly other operations. In order to determine the impact of the proposed formulations on the preprocessing phase of BB, we turn off all heuristics and allow CPLEX to process only the root node of the BB tree. After processing the root node, we compute the percent difference between the obtained dual bound and the LP bound obtained in the second of our previous tests described at the start of Section 6. The larger the percent difference, the better, as this indicates that the reductions performed resulted in a tighter relaxation. Results for this test are presented in Table 4 where, for each objective type and each formulation, we report the fraction of instances for which the dual bound obtained after preprocessing was better, i.e., lower, than the dual bound obtained using N.N, and the average absolute and percent differences between the dual bounds obtained before and after preprocessing. From Table 4 we observe that preprocessing provides improvements to the dual bound in all cases, but that, on average, improvements are greater for problems with cost objectives than profit objectives. Additionally, the dual bounds obtained after preprocessing are significantly better, on average, for our proposed reformulations than for the original formulation.

Table 4: Testing the impact of the proposed formulations on the preprocessing phase of BB

	Fraction Better Than N.N	Avg Absolute Difference	Avg Percent Difference
		Cost	
N.N	–	7.0	46.78%
N.BIA	103/115	14.1	62.95%
N.BIJA	105/115	15.3	69.40%
N.BIJTA	105/115	15.4	79.99%
		Profit	
N.N	–	5.7	2.22%
N.BIA	98/115	13.9	5.35%
N.BIJA	96/115	15.2	5.87%
N.BIJTA	96/115	15.5	6.01%

6.2. Heuristics

We now consider the impact of the proposed formulations on the heuristics employed by CPLEX. In order to test this, we turn off all cut generation and again allow CPLEX to process only the root node. After processing the root node, we record whether or not the heuristics were able to identify an integer feasible solution, which we henceforth refer to as an incumbent. For each instance, formulation pair for which an incumbent was successfully identified by the heuristics, we also record the optimality gap after processing the root node. Additionally, for each objective type, the fraction of instances for which the optimality gap found for each reformulation is better, i.e., smaller, than that of N.N is recorded. Results of this test are summarized in Table 5. We note that in Table 5 we report modified optimality gaps in a similar fashion to the modified values we reported earlier in Tables 1, 2, and 3. Specifically, we report in gray the optimality gap divided by the fraction of instances for which an incumbent solution was found. The motivation for the inclusion of this metric is that, in some sense, it penalizes each formulation proportionately to the number of instances for which the heuristics were not able to identify an incumbent.

It is interesting to recognize from Table 5 that in most cases the incumbents obtained from the reformulations proposed herein are not as high quality as those obtained from the original formulation. On one hand, this is surprising because, as seen in Section 4, using these reformulations generally results in an overall reduction in the CPU

Table 5: Testing the impact of the proposed formulations on the BB heuristic

	Fraction Incumbent Found	Fraction Better Than N.N	Avg Optimality Gap	
		Cost		
N.N	99/115	–	42.9%	(49.8)
N.BIA	90/115	22/115	44.9%	(57.4)
N.BIJA	92/115	18/115	44.6%	(55.8)
N.BIJTA	90/115	18/115	45.0%	(57.5)
		Profit		
N.N	101/115	–	44.7%	(50.9)
N.BIA	99/115	42/115	84.2%	(97.8)
N.BIJA	99/115	39/115	82.1%	(95.4)
N.BIJTA	99/115	41/115	78.7%	(91.4)

(Gray): (Average Optimality Gap) ÷ (Fraction Incumbent Found)

time required to solve instances to optimality and it is well known that the overall solution time of BB is often highly dependent on the ability of an optimizer to obtain quality incumbent solutions quickly. On the other hand, though, there are reasons that this result is to be expected. For one, we know that each reformulation involves the addition of several new integer variables and several new constraints to $\mathcal{P}$ and so, as these changes increase the size and complexity of each instance, it does make sense that this might have a negative impact on the performance of a heuristic. While we do not study the impact of doing so in this work, we note that in the future it may be interesting to develop an implementation in which, when $\mathcal{P}$ is reformulated using one of our proposed reformulations, a heuristic is applied to the unmodified version of $\mathcal{P}$ and the obtained incumbents are mapped to the reformulated problem using the appropriate subset of Equations (4)–(13).

6.3. Branching

We next study the branching phase of BB. In order to determine the impact of our proposed formulations on this phase of the BB procedure, we compare the number of nodes required to solve each instance. We note that for this test both cut generation and heuristics are turned back on, because without these features the performance of BB dramatically suffers. As a result, strictly speaking, the strong performance of our proposed reformulations displayed in the following data should not be interpreted as being due solely to differences in branching, but rather differences in the BB process as a whole. However, as the primary metric we report for this test is the number of nodes required to solve each instance, significant differences in these numbers can still be attributed primarily to the benefits of branching on record keeping variables. Results for this test are displayed in Table 6. The first piece of information recorded in Table 6 is the fraction of instances solved in under five hours. The remaining data is displayed for two specific subsets of instances: (i)

Table 6: Testing the impact of the proposed formulations on the branching phase of BB

	Fraction Solved	Avg Rel Nodes†	Avg Rel Nodes‡
		Cost	
		(46 instances)	(22 instances)
N.N	69/115	1.00	1.0000
N.BIA	95/115	0.46	0.0668
N.BIJA	101/115	0.30	0.0047
N.BIJTA	99/115	0.24	0.0604
		Profit	
		(16 instances)	(27 instances)
N.N	24/115	1.00	1.0000
N.BIA	62/115	0.46	0.0797
N.BIJA	78/115	0.28	0.0031
N.BIJTA	75/115	0.18	0.0038

†: Instances solved in < 5 hours by all formulations above, and N.N not solved at the root node

‡: Instances solved in > 5 hours by N.N and < 5 hours by all others

those that all considered formulations were able to solve in under five hours, but that were not solved by N.N at the root node, and (ii) those that N.BIA, N.BIJA, and N.BIJTA were able to solve in under five hours, but N.N was not. For each subset, we report average values for relative numbers of nodes calculated as

$$\text{Relative \# Nodes For Instance } \ell, \text{ Formulation } k = \frac{\text{Nodes to Solve Instance } \ell \text{ Using Formulation } k}{\text{Nodes to Solve Instance } \ell \text{ Using N.N}}$$

for the former subset, and

$$\text{Relative \# Nodes For Instance } \ell, \text{ Formulation } k = \frac{\text{Nodes to Solve Instance } \ell \text{ Using Formulation } k}{\text{Nodes Explored in 5 hrs by N.N for Instance } \ell}$$

for the latter. As can be seen from Table 6, all the considered reformulations greatly reduce the number of nodes required to solve instances of $\mathcal{P}$.

We end our discussion on the branching phase of BB by noting that our presented results are, again, not surprising. In fact, branching on record keeping variables can be viewed as a type of *constraint branching* as introduced by Ryan and Foster [47] and later generalized by Appleget and Wood [48], which has been shown to be useful in many other applications of MILP.

6.4. Further Analysis

Recall from the work done so far in this section that preprocessing and branching have been identified as the main phases of BB during which CPLEX is able to exploit certain features of our proposed reformulations to reduce solution time. We now seek to answer the following two questions:

1. Can we gain a deeper insight into what specific problem features CPLEX is exploiting to improve performance?
2. Are the results we observe unique to CPLEX, or do other MILP solvers display similar results?

In pursuit of answering Question 1, we take a deeper look at the reductions made by CPLEX when solving one of the instances we considered in our previous tests. Specifically, we consider the network referred to as "network 3" by Velez and Maravelias [39], which was originally presented in [44], and we employ the cost objective. The STN representation of this network is given in Figure 5, and we note that the optimal objective value of the problem is 1145. To conduct our analysis, we employ the CPLEX C API, together with a solve callback, in order to access the modified MILP problem that CPLEX actually solves at each node of the BB tree. We then access this problem for the root node once it has been fully processed. In this way we are able to see the cuts that CPLEX has generated (and retained) as well as any variable bounds that have

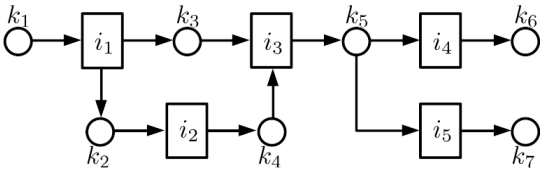


Figure 5: STN – Network from Velez et al. [44]

Task-unit mapping: $\mathbf{J}_{i_1} = \mathbf{J}_{i_2} = \{j_1\}$; $\mathbf{J}_{i_3} = \{j_2\}$; $\mathbf{J}_{i_4} = \{j_3\}$; $\mathbf{J}_{i_5} = \{j_4\}$.

been tightened prior to the start of the branching phase of BB. For this instance, when using the N.N formulation, CPLEX adds 25 cuts that are retained throughout the processing of the root node. We next solve the LP relaxation of the obtained problem, which gives an objective value of 1133.32. We note that at this solution, 15 of the abovementioned 25 cuts are binding. Similarly, when using the N.BIJTA formulation, CPLEX adds 36 cuts that are retained, 20 of which are binding at the LP solution. More interestingly, though, CPLEX is also able to generate the following bounds on record keeping variables that are binding at the LP solution:

$$N_{i_1,j_1} = 25$$

$$N_{i_3,j_2} = 15$$

$$N_{i_1} \geq 25$$

$$N_{i_2} \geq 7$$

$$N_{i_4} \geq 14$$

$$N_{i_5} \geq 11$$

$$N_{j_1} \geq 32$$

$$N_{j_2} \geq 15$$

It is important to point out here that the objective value of this LP is 1145, which shows that CPLEX has fully tightened the dual bound. It is also interesting to note that if we add constraints enforcing these same bounds to the LP obtained after processing the root node of the N.N formulation, the optimal value improves to 1145. This serves as evidence that the bound tightening phase of CPLEX's solution procedure significantly contributes to the reductions in solution time that are obtained when using our proposed reformulations.

In pursuit of answering Question 2, we select 18 of the instances considered in our previous tests for which CPLEX displayed an extreme difference in performance using formulation N.N as opposed to N.BIJTA, and we compare the performance of three different MILP solvers on the selected instances using these two formulations. Specifically, we employ the commercial solvers CPLEX 22.1.1 [49] and Gurobi 10.0.0 [50], as well as the open-source solver HiGHS 1.5.1 [51, 52], all via GAMS 42.4.0. The tests were run on a Dell Latitude 7420 with a 2.60GHz Intel i5-1145G7 processor and 16GB of RAM running Linux Mint 20.3. We note that the reason for the discrepancy in software used here compared to earlier parts of this work is our desire to employ the HiGHS solver for these tests, which is not available in the GAMS version used for our prior tests. We set a one hour time limit for all solvers for this test, the results of which are given in Table 7. We note that for the instance identifiers given in Table 7, "V" indicates an instance from [39], "M" indicates an instance from `minlp.org`, and "R" indicates a randomly generated instance. Furthermore, for instances from [39] the value given indicates the number assigned to the associated network in that work, for instances from `minlp.org` the value given indicates the horizon length, and for randomly generated instances the three underscore delimited values indicate $|\mathbf{I}|$, $|\mathbf{J}|$, and $|\mathbf{K}|$, respectively. Interestingly,

Table 7: Comparing MILP solver performance on selected instances

		CPLEX				Gurobi						HiGHS			
		N.N		N.BIJTA		N.N		N.BIJTA†		N.BIJTA‡		N.N		N.BIJTA	
Instance	Obj	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap
V2	Profit	3600	3.32	2.4	0	3600	3.86	3600	3.43	3600	3.87	3600	3.62	127.6	0
V3	Cost	3600	0.65	0.0	0	2.0	0	3.3	0	1.0	0	523.5	0	59.9	0
V4	Cost	3600	0.46	2.4	0	3.5	0	3.1	0	4.7	0	1034.1	0	12.6	0
V4	Profit	3600	0.21	6.1	0	3600	0.12	3600	0.21	3600	0.12	573.8	0	60.9	0
M32	Profit	3600	0.31	246.9	0	3600	0.50	3600	0.54	4.3	0	3600	1.16	166.0	0
M40	Cost	1175.4	0	0.7	0	76.3	0	24.1	0	0.6	0	3600	0.77	5.0	0
M44	Profit	3600	1.63	544.6	0	3600	0.79	3600	1.02	206.3	0	3600	2.74	1543.4	0
R5_3_6	Profit	402.7	0	3.7	0	5.3	0	27.0	0	0.8	0	3600	0.26	8.3	0
R5_3_7	Profit	3600	5.88	30.5	0	3600	1.75	3600	2.31	3600	1.11	3600	3.73	83.2	0
R5_4_5	Profit	3600	0.05	0.7	0	3.7	0	1.4	0	1.9	0	1843.3	0	15.0	0
R6_5_7	Profit	3600	0.36	8.0	0	3600	0.36	32.0	0	2.9	0	1986.7	0	39.0	0
R8_5_8	Cost	1429.0	0	13.1	0	21.5	0	926.9	0	5.1	0	3600	4.14	27.3	0
R8_11_8	Cost	3600	4.01	26.7	0	3600	5.64	3600	4.19	7.6	0	3600	12.74	1346.7	0
R9_11_7	Profit	3600	0.39	3600	0.02	3600	0.25	3600	0.42	3600	0.01	3600	0.48	1060.3	0
R10_8_11	Profit	3600	1.57	3600	0.01	3600	1.04	3600	1.29	3600	0.15	3600	1.72	3579.0	0
R11_15_13	Cost	3600	16.40	26.4	0	534.5	0	2949.2	0	59.4	0	3600	26.65	236.3	0
R12_10_11	Profit	3600	0.18	1169.7	0	94.3	0	153.0	0	51.6	0	3600	1.07	372.7	0
R13_26_12	Cost	3600	18.75	90.9	0	3600	12.90	3600	11.90	3600	5.32	3600	28.40	667.3	0

†: Gurobi not preceded by bound tightening

‡: Gurobi preceded by bound tightening

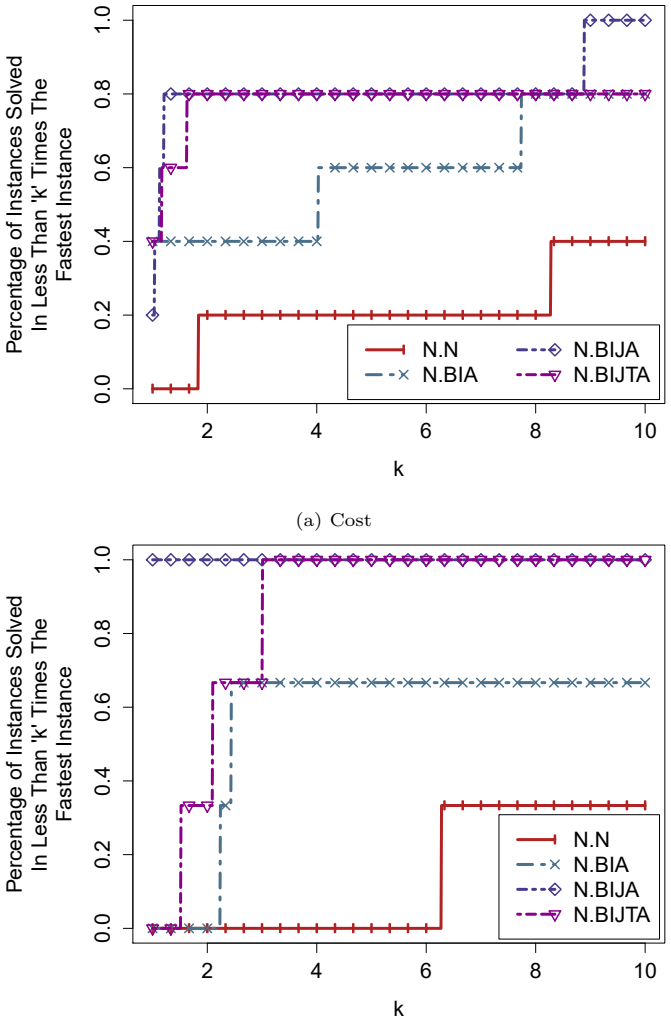
from Table 7 we observe a very similar pattern of performance between CPLEX and HiGHS, but not Gurobi. In studying the log files from our tests, we find it likely that this difference in performance is caused by Gurobi substituting all record keeping variables out of the model prior to running its probing and/or bound tightening schemes. To test whether or not this is the case and help determine if Gurobi's performance could be improved by exploiting valid bounds on record keeping variables, we reran the N.BIJTA portion of our tests with Gurobi, but this time we tightened the bounds on each record keeping variable *prior* to passing the problem to Gurobi. Assume that N^* represents an arbitrary record keeping variable. The approach we employ for computing a tight lower bound ℓ^* on N^* consists of solving the LP relaxation of problem $\mathcal{P}$ with $f(\cdot) = N^*$ and setting ℓ^* to be the ceiling of the optimal objective value. Similarly, we compute a tight upper bound u^* on N^* by solving the LP relaxation of problem $\mathcal{P}$ with $f(\cdot) = -N^*$ and setting u^* to be the floor of the negative of the optimal objective value. Results for our tests without bound tightening are indicated in Table 7 by the symbol † and the tests with bound tightening are indicated

by the symbol ‡. Additionally, we note that the solution time reported for the tests with bound tightening do not include time spent employing the bound tightening procedure. As can be seen in Table 7, in most cases Gurobi's performance significantly improves when provided tighter bounds for the record keeping variables.

From the tests conducted in this section, we are now quite confident that the main phases of BB in which information about record keeping variables can be exploited in order to improve the performance of a MILP solver are preprocessing (especially probing/bound tightening) and branching. Moreover, we also conclude that this improvement in performance is not unique to CPLEX, though we do note that not all solvers are equally able to achieve these improvements, seemingly due to the order in which certain presolve and/or preprocessing operations are carried out.

7. Utility of Proposed Reformulations for Other Problem Classes

Given the reductions in solution time that we observed when record keeping variables are added to $\mathcal{P}$, we now analyze the impact of incorporating record keeping variables when solving more complicated classes of problems. For this purpose we modify the instances from `minlp.org` that we utilized throughout the earlier sections of this work. These instances are provided in such a way that the user is able to employ one of two sets of modifications: (i) the incorporation of restrictions based on the availability of utility resources, or (ii) the incorporation of variable processing times based on batch size. We employ both modifications and refer to the first set of problems as UTL instances and the second as VPT instances. We utilize the same objective types mentioned earlier, i.e., cost, and profit.



(a) Cost

(b) Profit

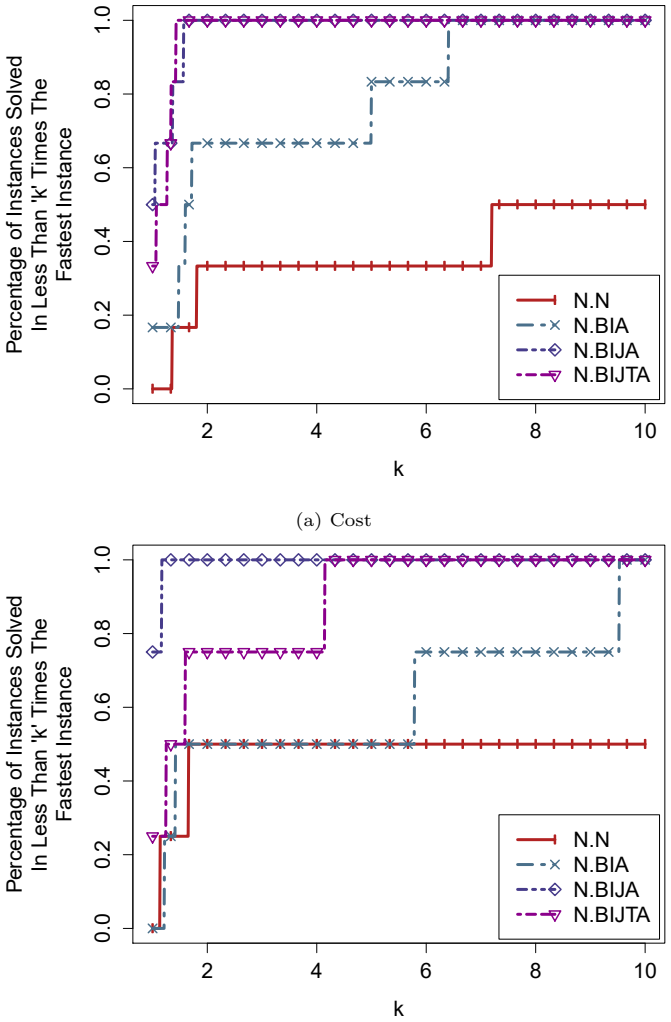
Figure 6: Performance profiles of CPU time (s) for selected formulations on UTL problems – Instances considered are those for which all formulations solved the instance in 5hrs

Results for the UTL instances are presented in Figure 6 and Table 8 and results for the VPT instances are presented in Figure 7 and Table 9. Interestingly, the

Table 8: Results for UTL instances for which at least one formulation failed to solve the instance in 5hrs and at least one formulation successfully solved the instance in 5hrs

	Fraction Solved	Avg Rel Time		Avg Rel Gap	
		Cost			
N.N	1/2	13.0	(26.0)	1.0	(2.0)
N.BIA	1/2	1.8	(3.5)	1.0	(2.0)
N.BIJA	2/2	2.1	(2.1)		–
N.BIJTA	2/2	2.8	(2.8)		–
		Profit			
N.N	0/3		–	1.0	(∞)
N.BIA	3/3	6.9	(6.9)		–
N.BIJA	3/3	3.0	(3.0)		–
N.BIJTA	3/3	13.0	(13.0)		–

(Gray): (Average Relative Value) ÷ (Fraction Solved)



(a) Cost

(b) Profit

Figure 7: Performance profiles of CPU time (s) for selected formulations on VPT problems – Instances considered are those for which all formulations solved the instance in 5hrs

Table 9: Results for VPT instances for which at least one formulation failed to solve the instance in 5hrs and at least one formulation successfully solved the instance in 5hrs

	Fraction Solved	Avg Rel Time		Avg Rel Gap	
			Cost		
N.N	0/1		–	1.0	(∞)
N.BIA	1/1	2.4	(2.4)		–
N.BIJA	1/1	1.0	(1.0)		–
N.BIJTA	1/1	2.5	(2.5)		–
			Profit		
N.N	0/3		–	4.9	(∞)
N.BIA	2/3	5.4	(8.1)	1.0	(1.5)
N.BIJA	3/3	3.1	(3.1)		–
N.BIJTA	2/3	3.1	(4.6)	434.8	(652.2)

(Gray): (Average Relative Value) ÷ (Fraction Solved)

results for both UTL and VPT instances are quite similar. In observing Figures 6 and 7 and Tables 8 and 9, we observe the strongest performance from N.BIJA and N.BIJTA, slightly weaker performance from N.BIA, and relatively poor performance from N.N.

8. Conclusion

We have introduced the concept of a *record keeping variable* as well as several ways to formulate certain classes of production scheduling problems as discrete-time based mixed-integer linear programs that incorporate record keeping variables. Moreover, we have shown empirically that the presence of record keeping variables in a problem's formulation can drastically reduce the CPU time required to solve the problem. In all, we considered 230 instances of chemical production scheduling problems (excluding the UTL and VPT variants). Of these, there were 27 instances that both our proposed method N.BIJA and method N.N (representative of employing default CPLEX to solve the base formulation of problem $\mathcal{P}$) were able to solve in under 5 hours and that at least one of these approaches was unable to solve in 3 minutes. For these instances, the CPU time used by our approach N.BIJA was 2.79% of that used by N.N, on average. Additionally, N.BIJA was able to solve 83 instances in under 5 hours that N.N was not. Similarly, there were 25 instances that both our proposed method N.BIJA and method N.B (representative of employing default CPLEX to solve the most promising formulation proposed in [39]) were able to solve in under 5

hours and that at least one of these approaches was unable to solve in 3 minutes. For these instances, the CPU time used by our approach N.BIJA was 3.25% of that used by N.B, on average. Additionally, N.BIJA was able to solve 70 instances in under 5 hours that N.B was not.

Later we showed that prioritizing branching on record keeping variables over other integer variables works well in certain cases, particularly when N_{ij}, N_i, N_j and N are simultaneously included in the model and the overall solution time is estimated to be less than 5 hours. In addition to showing the impact of our proposed reformulations on the CPU time required to solve instances, we sought to determine *why* the improvements that we observed were obtained by exploring the impact of our proposed reformulations on the preprocessing, heuristic, and branching phases of the branch-and-bound procedure. Our results showed that the incorporation of record keeping variables in a problem's formulation can result in significant improvements in the quality of the variable bounds and/or cuts generated and the number of nodes required to solve the problem. Interestingly, we found that similar improvements were not achieved in generating quality heuristic solutions. Finally, we showed that the incorporation of record keeping variables into the formulation of more complex classes of scheduling problems can also result in significant reduction in solution time. We note that this last point is of particular importance as it provides evidence that one can still expect to achieve improved performance by applying the techniques proposed herein to process scheduling problems whose formulations account for specialized, application specific process characteristics.

References

[1] I. Harjunkoski, C. T. Maravelias, P. Bongers, P. M. Castro, S. Engell, I. E. Grossmann, J. Hooker, C. Méndez, G. Sand, J. Wassick, Scope for industrial applications of production scheduling models and solution methods, Computers & Chemical Engineering 62 (2014) 161–193.

[2] G. P. Georgiadis, A. P. Elekidis, M. C. Georgiadis, Optimization-based scheduling for the process industries: from theory to real-life industrial applications, Processes 7 (2019) 438.

[3] O. Avalos-Rosales, A. Alvarez, F. Angel-Bello, A reformulation for the problem of scheduling unrelated parallel machines with sequence and machine dependent setup times, in: Proceedings of the International Conference on Automated Planning and Scheduling, volume 23, 2013, pp. 278–282.

[4] S. Velez, Y. Dong, C. T. Maravelias, Changeover formulations for discrete-time mixed-integer programming scheduling models, European Journal of Operational Research 260 (2017) 949–963.

[5] D. C. Cafaro, I. E. Grossmann, Strengthening discrete-time scheduling formulations by introducing the concept of campaigns, Computers & Chemical Engineering 143 (2020) 107101.

[6] A. Kramer, M. Iori, P. Lacomme, Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization, European Journal of Operational Research 289 (2021) 825–840.

[7] J. Berkhout, E. Pauwels, R. van der Mei, J. Stolze, S. Broersen, Short-term production scheduling with non-triangular sequence-dependent setup times and shifting production bottlenecks, International Journal of Production Research 59 (2021) 727–751.

[8] P. M. Castro, Optimal scheduling of a multiproduct batch

chemical plant with preemptive changeover tasks, Computers & Chemical Engineering 162 (2022) 107818.

[9] P. M. Castro, I. Harjunkoski, I. E. Grossmann, Discrete and continuous-time formulations for dealing with break periods: Preemptive and non-preemptive scheduling, European Journal of Operational Research 278 (2019) 563–577.

[10] Y. Wang, X. Jin, S. Lu, Enhanced discrete time formulation for a short-term batch process scheduling problem with utility constraints, Industrial & Engineering Chemistry Research 58 (2019) 14559–14568.

[11] D. Gupta, C. T. Maravelias, J. M. Wassick, From rescheduling to online scheduling, Chemical Engineering Research and Design 116 (2016) 83–97.

[12] D. Gupta, C. T. Maravelias, On the design of online production scheduling algorithms, Computers & Chemical Engineering 129 (2019) 106517.

[13] R. D. McAllister, J. B. Rawlings, C. T. Maravelias, The inherent robustness of closed-loop scheduling, Computers & Chemical Engineering 159 (2022) 107678.

[14] C. A. Floudas, X. Lin, Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review, Computers & Chemical Engineering 28 (2004) 2109–2129.

[15] A. Sundaramoorthy, C. T. Maravelias, Computational study of network-based mixed-integer programming approaches for chemical production scheduling, Industrial & Engineering Chemistry Research 50 (2011) 5023–5040.

[16] Z.-L. Chen, W. B. Powell, Solving parallel machine scheduling problems by column generation, INFORMS Journal on Computing 11 (1999) 78–94.

[17] Z.-L. Chen, W. B. Powell, Exact algorithms for scheduling multiple families of jobs on parallel machines, Naval Research Logistics (NRL) 50 (2003) 823–840.

[18] S. Gélinas, F. Soumis, Dantzig-wolfe decomposition for job shop scheduling, in: Column generation, Springer, 2005, pp. 271–302.

[19] M. J. P. Lopes, J. V. de Carvalho, A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times, European journal of operational research 176 (2007) 1508–1527.

[20] J. Van den Akker, J. Hoogeveen, J. W. van Kempen, Using column generation to solve parallel machine scheduling problems with minmax objective functions, Journal of Scheduling 15 (2012) 801–810.

[21] A. Ghoniem, F. Farhadi, A column generation approach for aircraft sequencing problems: a computational study, Journal of the Operational Research Society 66 (2015) 1717–1729.

[22] X. Xiong, P. Zhou, Y. Yin, T. Cheng, D. Li, An exact branch-and-price algorithm for multitasking scheduling on unrelated parallel machines, Naval Research Logistics (NRL) 66 (2019) 502–516.

[23] V. Jain, I. E. Grossmann, Algorithms for hybrid milp/cp models for a class of optimization problems, INFORMS Journal on computing 13 (2001) 258–276.

[24] P. Baptiste, C. Le Pape, W. Nuijten, Constraint-based scheduling: applying constraint programming to scheduling problems, volume 39, Springer Science & Business Media, 2001.

[25] A. Bockmayr, N. Pisaruk, Detecting infeasibility and generating cuts for mip using cp, in: 5th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems-CPAIOR'03, 2003, pp. 11–p.

[26] G. El Khayat, A. Langevin, D. Riopel, Integrated production and material handling scheduling using mathematical programming and constraint programming, European Journal of Operational Research 175 (2006) 1818–1832.

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

[28] L. Zeballos, O. Quiroga, G. P. Henning, A constraint programming model for the scheduling of flexible manufacturing systems with machine and tool limitations, Engineering Applications of Artificial Intelligence 23 (2010) 229–248.

[29] E. B. Edis, I. Ozkarahan, A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions, Engineering Optimization 43 (2011) 135–157.

[30] A. M. Ham, Integrated scheduling of m-truck, m-drone, and m-depot constrained by time-window, drop-pickup, and m-visit using constraint programming, Transportation Research Part C: Emerging Technologies 91 (2018) 1–14.

[31] R. Gedik, D. Kalathia, G. Egilmez, E. Kirac, A constraint programming approach for solving unrelated parallel machine scheduling problem, Computers & Industrial Engineering 121 (2018) 139–149.

[32] L. Meng, C. Zhang, Y. Ren, B. Zhang, C. Lv, Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem, Computers & Industrial Engineering 142 (2020) 106347.

[33] M. Awad, K. Mulrennan, J. Donovan, R. Macpherson, D. Tormey, A constraint programming model for makespan minimisation in batch manufacturing pharmaceutical facilities, Computers & Chemical Engineering 156 (2022) 107565.

[34] P. M. Castro, I. E. Grossmann, Generalized disjunctive programming as a systematic modeling framework to derive scheduling formulations, Industrial & Engineering Chemistry Research 51 (2012) 5781–5792.

[35] P. M. Castro, I. E. Grossmann, P. Veldhuizen, D. Esplin, Optimal maintenance scheduling of a gas engine power plant using generalized disjunctive programming, AIChE journal 60 (2014) 2083–2097.

[36] P. M. Castro, I. Marques, Operating room scheduling with generalized disjunctive programming, Computers & Operations Research 64 (2015) 262–273.

[37] H. Mostafaei, P. M. Castro, A. Ghaffari-Hadigheh, Short-term scheduling of multiple source pipelines with simultaneous injections and deliveries, Computers & Operations Research 73 (2016) 27–42.

[38] O. Wu, G. Dalle Ave, I. Harjunkoski, L. Imsland, A rolling horizon approach for scheduling of multiproduct batch production and maintenance using generalized disjunctive programming models, Computers & Chemical Engineering 148 (2021) 107268.

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

[40] E. Kondili, C. C. Pantelides, R. W. Sargent, A general algorithm for short-term scheduling of batch operations—i. milp formulation, Computers & Chemical Engineering 17 (1993) 211–227.

[41] N. Shah, C. C. Pantelides, R. W. H. Sargent, A general algorithm for short-term scheduling of batch operations. 2. computational issues, Computers & Chemical Engineering 17 (1993) 229–244.

[42] L. G. Papageorgiou, C. C. Pantelides, Optimal campaign planning/scheduling of multipurpose batch/semicontinuous plants. 1. mathematical formulation, Industrial & engineering chemistry research 35 (1996) 488–509.

[43] C. T. Maravelias, Mixed-time representation for state-task network models, Industrial & engineering chemistry research 44 (2005) 9129–9145.

[44] S. Velez, A. Sundaramoorthy, C. T. Maravelias, Valid inequalities based on demand propagation for chemical production scheduling mip models, AIChE Journal 59 (2013) 872–887.

[45] IBM-ILOG, CPLEX Optimization Studio 20.1 User's Manual, IBM, 2020. Available from `https://www.ibm.com/docs/en/icos/20.1.0?topic=cplex-users-manual`.

[46] E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, Mathematical programming 91 (2002) 201–213.

[47] D. M. Ryan, B. A. Foster, An integer programming approach to scheduling, Computer scheduling of public transport urban passenger vehicle and crew scheduling (1981) 269–280.

[48] J. A. Appleget, R. K. Wood, Explicit-constraint branching for

solving mixed-integer programs, in: Computing Tools for Modeling, Optimization and Simulation, Springer, 2000, pp. 245–261.

[49] IBM-ILOG, CPLEX Optimization Studio 22.1.1 User's Manual, IBM, 2022. Available from https://www.ibm.com/docs/en/icos/22.1.1?topic=optimizers-users-manual-cplex.

[50] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

[51] HiGHS, Documentation, HiGHS, 2022. Available from https://highs.dev/#docs.

[52] Q. Huangfu, J. J. Hall, Parallelizing the dual revised simplex method, Mathematical Programming Computation 10 (2018) 119–142.