Greedy Algorithm for Scheduling Batch Plants with Sequence-Dependent Changeovers

Pedro M. Castro
Laboratório Nacional de Energia e Geologia, Unidade de Modelação e Optimização de Sistemas Energéticos, 1649-038 Lisboa, Portugal

Iiro Harjunkoski
ABB Corporate Research Center, Wallstadter Str. 59, 68526 Ladenburg, Germany

Ignacio E. Grossmann
Dept. of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213

DOI 10.1002/aic.12261
Published online April 22, 2010 in Wiley Online Library (wileyonlinelibrary.com).

This article presents a new algorithm for scheduling multistage batch plants with a large number of orders and sequence-dependent changeovers. Such problems are either intractable when solved with full-space approaches or poor solutions result. We use decomposition on the entire set of orders and derive the complete schedule in several iterations, by inserting a couple of orders at a time. The key idea is to allow for partial rescheduling without altering the main decisions in terms of unit assignments and sequencing (linked to the binary variables) so that the combinatorial complexity is kept at a manageable level. The algorithm has been implemented for three alternative continuous-time mixed integer linear programing models and tested through the solution of 10 example problems for different decomposition settings. The results show that an industrial-size scheduling problem with 50 orders, 17 units distributed over six stages can effectively be solved in roughly 6 min of computational time.
 AIChE J, 57: 373–387, 2011
Keywords: optimization, mathematical modeling, continuous-time, resource-task network

Introduction

Modern enterprises of today are complex global networks of multiple business units and functions operating in a very dynamic environment. Long-term survival in the global marketplace can only be ensured if companies optimize the various functions within their supply chain, a process that has been named[1,2] as enterprise-wide optimization. The optimal operation of manufacturing plants is naturally involved and includes different levels of decisions such as planning, scheduling, and control.[3] The term scheduling is used in various contexts and can be closely interlinked with many related optimization problems, such as maintenance planning, energy and inventory optimization, and cutting-stock problems. It basically involves the allocation of production resources to tasks over a scheduling horizon between days and a few weeks.

Despite the substantial developments in the last couple of decades in the process systems engineering community[4,5] with the appearance of unified frameworks for the systematic modeling of industrial processes and powerful mathematical models, scheduling tools are only slowly spreading to industry.[6,7] The success story reported by Wassick[6] involves a waste treatment network from Dow Chemical Company that was optimized using a discrete-time resource-task network model,[8] which has shown very high potential. Yet, there are cases where the required time granularity lies within seconds

Additional Supporting Information may be found in the online version of this article.

Correspondence concerning this article should be addressed to P. M. Castro at pedro.castro@ineti.pt.

and minutes, for which continuous-time models remain the only valid approach. Scheduling problems arising in process plants with equipment units subject to sequence-dependent changeovers that differ significantly from the processing times, and dealing with makespan minimization, are perhaps the best-known case.[9–13] However, the computational studies performed in these references clearly show that the scope of current state-of-the-art mathematical programing formulations is limited to relatively small problems.

To make full-space formulations more attractive for real-world applications consisting of hundreds of batches, dozens of equipment units, and long scheduling horizons, efforts have also been oriented toward systematic techniques that are able to maintain the number of simultaneous decisions at a reasonable level. The goal is no longer to guarantee optimality, but finding very good solutions rapidly, typically within a few minutes of computational time.

Much work exists in the arena of heuristic scheduling algorithms both in chemical engineering and operations research. For instance, in the context of multistage batch plants with a single processing unit per stage and all products being produced in the same order on each unit, Ku and Karimi[14] have proposed an approximate mixed integer linear program (MILP) algorithm. Using the product sequence given by a rapid access heuristic,[15] the number of binary variables was reduced by confining the actual position to a close neighborhood. The outcome was a 50% reduction in computational time with schedules within 1% of the optimal makespans obtained by the full-space formulation. The same authors[16] also proposed a generalized completion time algorithm for the same plant topology, a mix of storage policies and sequence-dependent changeovers. The complete schedule is derived one product at a time for a given sequence, and no rescheduling is allowed in later iterations. The problem of finding a good sequence for processes with tardiness penalties was then addressed.[17] Four different algorithms were proposed, one of them using simulated annealing, where the main idea is to change the position of one product at a time while keeping the relative position of other products unchanged.

The same concept of releasing a subset of jobs and inserting them back into the schedule has been applied by Roslöf et al.[18] to improve the solution from a paper-converting mill with a single processing unit. They used a continuous-time MILP model with sequencing variables (SVs) that can also be used to insert a new set of jobs and provided guidelines on how to adapt the algorithm for parallel units. In such plants, rescheduling may involve reordering and reallocation of jobs to units.[19]

The movement from full-space to decomposition methods working with a reduced set of decisions allows us to tackle larger problems, but there comes a point where we can no longer work with the entire set of jobs simultaneously. When switching from short-term to medium-term scheduling, a rolling-horizon[20] approach is typically used. A small part of the time horizon is scheduled in detail, while determining at the same time the planning decisions for the remaining part with an approximated model. The partial schedule is then fixed with the following iteration considering the subsequent portion in detail. The real challenge with this approach is finding an approximate model that leads to decisions that can actually be implemented in practice when considering the detailed model. For a multiproduct plant, the decisions involve selecting the products to be considered on a particular subhorizon. The approaches in Lin et al.[21] and Janak et al.[22] were successfully tested on a large-scale plant with over 80 equipment units including processing recipes of hundreds of products. The authors also proposed a rescheduling procedure within the same framework to provide an immediate response to unexpected events such as equipment breakdown or the addition of orders.[23]

The main drawback of decomposition approaches is that there is rarely an indication on how good the solutions really are. Optimality is naturally lost as a consequence of the reduced search space, but one should ideally lose as little as possible. Among the different types of continuous-time formulations, time grid–based models are strongly dependent on the specified number of slots in the grid, unlike their sequence variables counterparts. They have however a larger scope and can handle more complex network structures, so there is a strong incentive for improving them. The difficulty is that the minimum number of slots that guarantees optimality cannot accurately be predicted, which becomes a serious issue because a single increase in the number of slots typically results in a one order of magnitude increase in computational effort. Although one might be tempted to use the most conservative estimation on the required number of slots, equal to the number of orders, it is almost always impractical to consider such large values. Thus, a decomposition algorithm relying on a time grid–based continuous-time formulation should ensure that the minimum number of slots is considered per iteration, if good quality solutions need to be provided with few computational resources. This issue has been neglected by Janak et al.[22,23] when using the unit-specific model of Ierapetritou and Floudas[24] in their large-scale multipurpose plant study from BASF.

This article proposes a new algorithm for the short-term scheduling of multistage batch plants. It uses essentially the same ideas of Roslöf et al.[18] and Méndez and Cerdá,[19] which have been recently[25] implemented with a unit-specific formulation, together with a sequence-based continuous-time model. We consider the same plant layout, but now with equipment units subject to sequence-dependent changeovers between orders. The nontrivial extension results from the existence of two alternative, very efficient, unit-specific models[9] that either consider changeovers implicitly, with three-index binary variables, or explicitly, through four-index binaries with combined processing and changeover tasks. Interestingly, it is the simpler option that leads to major changes in the decomposition strategy because the former mathematical formulation requires processing tasks to occupy consecutive slots in the grid. With the latter model, the challenge is identifying which combined tasks are required for a particular unit on a given iteration and their possible slot assignments.

The other algorithmic features are kept. In particular, the uncertainty in the number of time slots is unveiled as part of the search strategy without compromising tractability of the mathematical problem. In other words, the decomposition algorithm will consider the minimum number of slots per unit/iteration that enables to account for all possibilities within the constrained scenario, thus reducing solution degeneracy and problem size. As a consequence, time grids associated to different units will typically grow at different

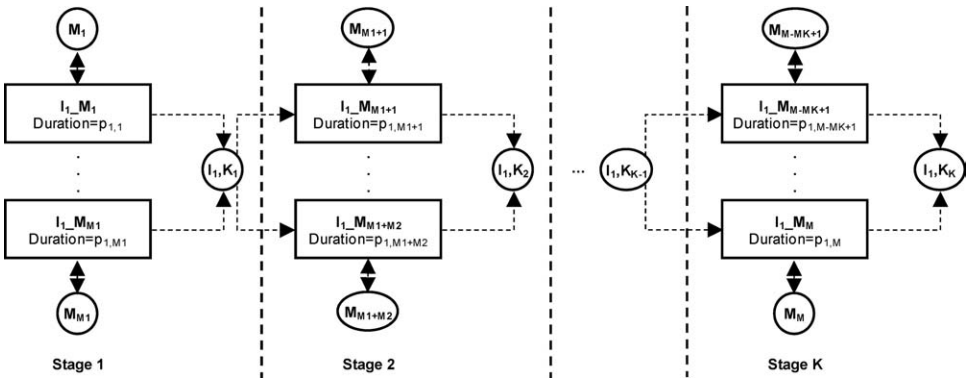

Figure 1. Schematic of a multistage multiproduct plant (dash arrows highlight that production is batch).

rates, which is a major breakthrough when compared with unit-specific full-space models that use the same number of slots for all time grids.

Besides the three alternative continuous-time formulations that can be used, the scheduling algorithm can be parameterized for the fast and efficient solution of problems of varying size. Seven small-size benchmark examples are used to validate the algorithm by measuring the optimality gap and differences in computational time compared with the corresponding full-space formulations for three distinct solution strategies. The method is applied to three industrial size instances, where the objective is to minimize the makespan. Note that in a real scheduling environment minimizing the makespan alone does not always make sense because of the continuity of production, distribution of end-customer due dates, and periods where the throughput does not need to be maximized. More common is to minimize the production costs (of which the makespan can be a component) or maximize the profit. However, from the computational point of view, makespan minimization is one of the most complex objectives and is therefore used in this study.

Problem Definition

Given a multistage, multiproduct plant with $k \in K$ processing stages, $i \in I$ product orders, and $m \in M$ units, the goal is to determine the assignment of orders to units and the sequence of orders at each unit so as to minimize the makespan. It is assumed that the orders with fixed processing times ($p_{i,m}$), release (r_i), and due dates (d_i) have been created following the decision of the number and size of batches to produce for each product. Batch identity is thus implicitly maintained throughout the process. Due dates are enforced as hard constraints, and there are sequence-dependent changeover times ($cl_{i,i',m}$). A particular equipment unit can handle all orders belonging to set I_m and belongs to a single stage, with set M_k defining the equipment at stage k. Some orders may skip certain production stages, which is defined by set I_k (orders processed in stage k). Unlimited intermediate storage and wait policies are assumed, and transfer times between units are considered to be negligible.

The process representation of a generic multistage multiproduct plant is given in Figure 1 for order I1, in the form of a resource-task network.[8] The plant resources (circles) are the equipment units and material states, whereas the processing tasks are represented as rectangles. The material state is directly associated to the stage where it is produced. Thus, processing tasks belonging to stage k consume material at state $k-1$ and produce material at state k.

Equipment units must be at appropriate cleaning states before processing tasks can be executed. Units must be at exactly one state at any given time, which will normally change throughout the scheduling horizon. For unit M1, the possible states are illustrated in Figure 2. In general, cleaning tasks (i,i',m) change the state from (i,m) into (i',m) so that the processing task of order i' can immediately follow in unit m. Note that a processing task does not change the equipment state.

Key Idea of Decomposition Approach

In large-scale scheduling problems, the number of orders greatly exceeds the number of equipment units. Because of

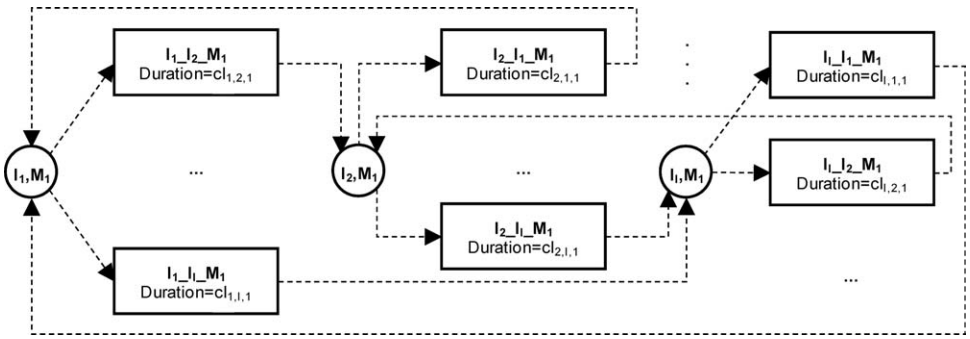

Figure 2. The possible cleaning tasks for a certain unit (M1) increase with the number of orders.

the high combinatorial complexity, such problems are intractable even by state-of-the-art full-space scheduling models and alternative approaches must be sought. The decomposition method proposed in this article reduces the complexity by scheduling a subset of the orders at a time.

Finding a schedule for a multistage plant with parallel units involves two decision levels: (i) assigning orders to units and (ii) sequencing orders on every unit. We follow this hierarchy to set different degrees of freedom for the orders. Those being considered for the first time can be assigned to all possible units and take any position in the sequence. In contrast, previously scheduled orders have significantly less freedom. Although the timing of events is allowed to change, orders cannot be reassigned to other units. Furthermore, their relative position in the sequence remains mostly unchanged.

Depending on the number of orders that are scheduled at a time (NOS), more and faster or fewer and slower iterations (set J) will be involved. Increasing NOS widens the feasible region up to that of the full-space model (NOS $= |I|$) so better solutions are likely to result if all generated mathematical problems can still be solved to optimality.

$$|J| = \lceil |I|/\mathrm{NOS} \rceil. \tag{1}$$

To select the sequence in which the orders are inserted into the schedule (set I_j^{iter} gives the orders being considered for the first time in iteration j), we will be relying on the increasing slack times heuristic (MST). Orders with a smaller time window (span_i) are thus scheduled first. We have chosen the MST heuristic as it performed better than the earliest due date heuristic in previous work.[25] Other heuristics can naturally be used, probably leading to significant differences in both solution quality and computational effort.

$$\mathrm{span}_i = d_i - r_i - \sum_{k \in K} \min_{\substack{m \in M_i \\ m \in M_k}} p_{i,m} \ \forall i \in I. \tag{2}$$

The decomposition algorithm has been developed to rely on two alternative time grid–based mathematical formulations that have shown similar performances in Castro et al.[9] The first, CT3I, handles changeover tasks implicitly giving rise to smaller problems due to the use of three-index binary variables (order, unit, and time slot). The second is tighter, CT4I, due to the explicit consideration of combined processing and changeover tasks and the use of four-index binaries (order, order, unit, and slot). Both rely on the concept of unit-specific time slots, with a single slot per order being enough to account for all valid alternatives. Note that with unit-specific slots, each processing unit keeps track of events taking place on its own time grid so index m can also be used to identify a particular grid.

One interesting aspect is that the decomposition algorithm no longer assumes the same number of slots per grid, contrary to full-space formulations that need to reduce the number of a priori decisions that may compromise global optimality. In fact, the time grids will grow between iterations and at different rates in terms of number of slots, between units. The number of slots to consider for a particular unit will be the minimum value that accounts for all valid alternatives within the constrained scenario and will depend on previous order assignments and NOS. More importantly, it will also be a function of the underlying model, with CT4I generating larger time grids and a higher number of decision variables. Counterintuitively, we will see that CT4I leads to a better performance.

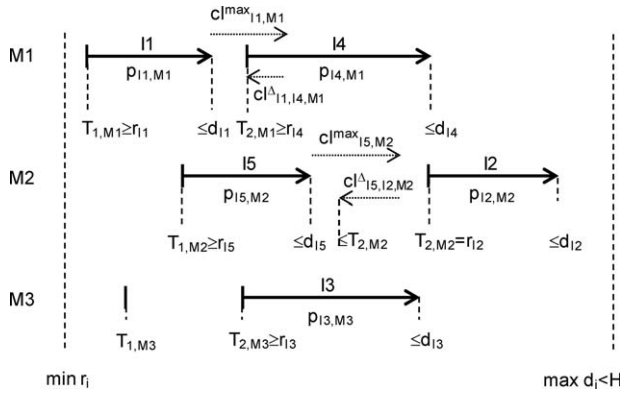

Figure 3. In formulation CT3I, changeovers are calculated through an indirect procedure ($|K| = 1$).

Three-index binary variables model (CT3I)

A simple single-stage example is given in Figure 3 to illustrate how CT3I accounts for the sequence-dependent changeover times. An indirect procedure is used that starts by calculating the maximum changeover times ($\mathrm{cl}_{i,m}^{\max}$) and the difference to the actual changeover ($\mathrm{cl}_{i,i',m}^{\Delta}$).

$$\mathrm{cl}_{i,m}^{\max} = \max_{i' \in I_m} \mathrm{cl}_{i,i',m} \ \forall m \in M, i \in I_m \tag{3}$$

$$\mathrm{cl}_{i,i',m}^{\Delta} = \mathrm{cl}_{i,m}^{\max} - \mathrm{cl}_{i,i',m} \ \forall m \in M, i, i' \in I_m. \tag{4}$$

Assigning order i to unit m brings a positive contribution to the slot duration, equal to $p_{i,m} + \mathrm{cl}_{i,m}^{\max}$, which considers the worst case scenario in terms of changeovers. To get the true changeover time, one must subtract $\mathrm{cl}_{i,i',m}^{\Delta}$ if task i' is to be executed at the next time interval (e.g., I1 and I4 in M1, I5 and I2 in M2). This procedure makes it more favorable for orders to be executed in consecutive intervals. Generality is ensured by not adding the maximum changeover term to the processing time of tasks executed in the last interval (see orders I4, I2, and I3). To reduce solution degeneracy, it is convenient to force orders to be assigned from the last to the first event point. Note in Figure 3 that order-unit-slot allocation is only possible if the starting time of slot t in unit m (variable $T_{t,m}$) is lower than the release date of order i.

The constructive scheduling formulation using CT3I as the underlying mathematical formulation is illustrated in Figure 4 for the case of one order first considered per iteration (NOS $= 1$) with an example consisting of four units (two per stage). Potential assignments are identified as unfilled boxes to distinguish from the actual decisions taken. As we are focusing on order-unit-slot allocation and to make the diagram compact, timing issues between slots of dissimilar units have been neglected. Thus, it may seem that an order starts to be processed in either M3 or M4 (stage two) before being completed in either M1 or M2 (stage 1) even though this cannot happen.

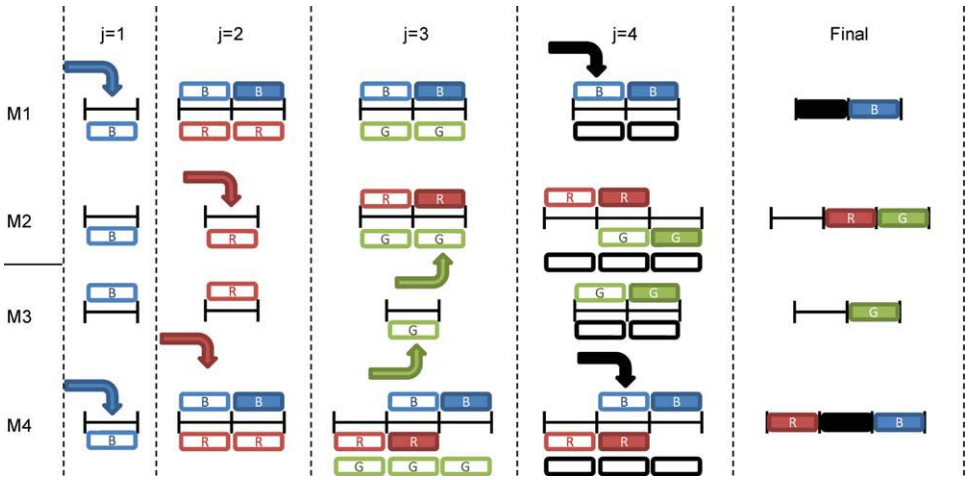

Figure 4. Illustration of constructive scheduling algorithm for model CT3I and one order at a time (NOS = 1).

[Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

In the first iteration ($j = 1$), it suffices to postulate one slot per grid to receive the blue order. Let us assume that the solver decides to assign the order to M1 and M4 (indicated by the arrows). In iteration $j = 2$, the time grids of such units feature two slots to account for the production of the newly considered red order. Units M2 and M3 remain with a single slot as they can no longer receive the previously assigned blue order. One can see the possible slot assignments in Figure 4, with filled rectangles showing previously scheduled orders. In particular, they are placed from right to left to reduce solution degeneracy.

In general, the number of slots to postulate for a given unit is equal to the number of already assigned orders plus the chosen NOS value. In the third iteration, there are three slots in M4 to accommodate the blue and red orders and eventually the new green order. Notice that not all slot assignments are allowed to avoid disturbing the relative sequence of the red and blue orders. Thus, the blue order cannot start before slot #2 neither can red be executed in slot #3. An additional black order is included to end the illustration with iteration $j = 4$.

The same concept can be applied to other values of parameter NOS. Figure 5 illustrates the outcome of the different iterations for NOS = 2 assuming the same final schedule. The points to highlight are as follows: (i) there are significantly more decisions to be made per iteration but half the number of iterations, thus leading to more complex but fewer mathematical problems; (ii) we can no longer

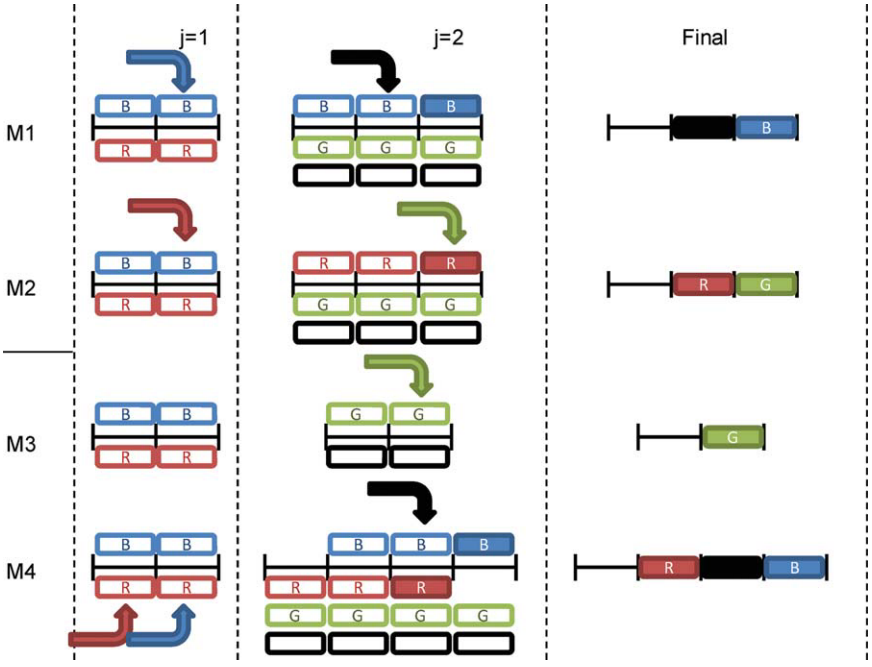

Figure 5. Illustration of constructive scheduling algorithm for model CT3I and NOS = 2.

[Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

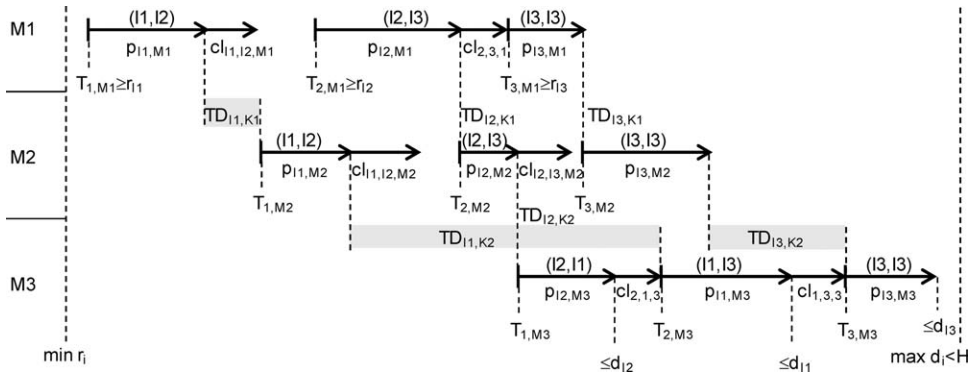

Figure 6. Formulation CT4I models changeovers explicitly through combined processing and changeover tasks ($|K| = 3$).

guarantee that the relative sequence of previously assigned orders is kept as soon as two orders get assigned to a unit (e.g., for M4 and $j = 2$, assigning blue to slot #2 and red to slot #3 makes it possible having blue before red, contrary to the outcome of $j = 1$).

Four-index model (CT4I)

Formulation CT4I explicitly accounts for changeovers through the definition of changeover tasks. In fact, they are merged with processing tasks to form combined processing and changeover tasks. The execution of combined task (i,i',m) comprises the processing time of order i plus the required changeover from i to i' so that unit m is ready for order i' to immediately follow. This aspect is illustrated in Figure 6, where it can be seen that processing in stage $k + 1$ can start right after finishing the processing part of the combined task in stage k (e.g., order I2 from unit M1 to M2 and also from M2 to M3). Although in the full-space model there are constraints[10] that reduce solution degeneracy by forcing the last task to be executed to feature a single-order index, the same cannot be done for the decomposition strategy. Thus, the last task to be executed on nonlimiting units (completion time does not affect the makespan) might be of type (i,i',m) with $i \neq i'$ and $cl_{i,i',m} \neq 0$.

The constructive scheduling algorithm for CT4I is illustrated in Figure 7 for NOS = 1. Combined tasks with a single-order index are represented with a single box, e.g. (blue, blue) in $j = 1$. For the others, two boxes are used. The first rectangle within the slot identifies the processing part, whereas the square identifies the changeover fraction. Note that the following processing task must be of the latter color.

The main conceptual difference when compared with CT3I is that now previously assigned orders can only be executed in a single slot. In general, and depending on its position in the sequence from the previous iteration ($pos_{i,m}$), the order will be assigned to slot number $pos_{i,m} \times (NOS + 1)$. This strategy has recently been proposed by Castro et al.[25] and was named fixed relative positioning. The novelty here is that because of the sequence-dependent changeovers there is more than one suitable task for previously assigned orders. Taking iteration $j = 2$ for units M1 and M4 as an example, the blue order is confined to the second time slot and the two potential tasks are (blue, blue) and (blue, red), making it possible to have the red order executed before or after blue, respectively.

Things become increasingly more interesting as we proceed through iterations. For $j = 3$ and M4, it is sufficient to consider (red, blue), (red, green), (blue, green), and (blue, blue) with respect to previously assigned orders. Note that all possible sequences are accounted for: (a) green–red–blue is achieved with (green, red) in slot #1, (red, blue) in #2, and (blue, blue) in #4; (b) red–green–blue with (red, green) in slot #2, (green, blue) in #3 and (blue, blue) in #4; (c) red–blue–green results from (red, blue), (blue, green) in slot #4 and (green, green) in #5.

The same principle can be applied to NOS = 2, see Figure 8. Nevertheless, some combined tasks of new orders, no

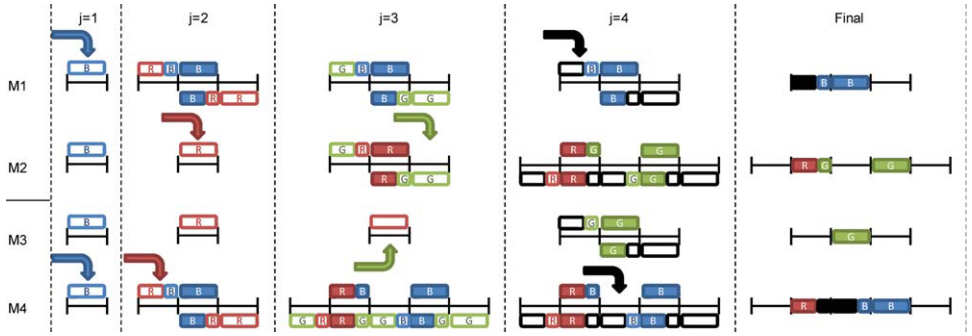

Figure 7. Illustration of constructive scheduling algorithm for model CT4I and NOS = 1.

[Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

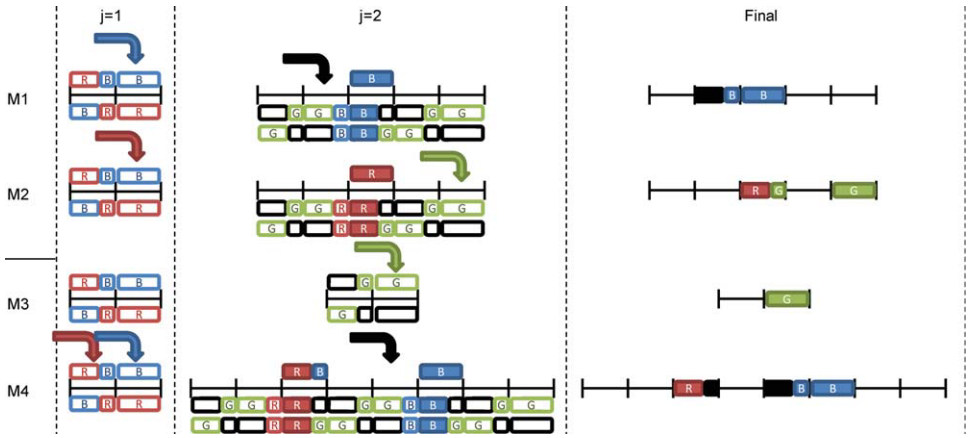

Figure 8. Illustration of constructive scheduling algorithm for model CT4I and NOS = 2.

[Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

longer have a single appropriate slot for their execution. Consider (black, green) in $j = 2$ as an example. In unit M1 it may be executed in slots #1 and #4 as it is possible to have black–green either before or after blue. In unit M4 it appears three times. Note that the black–red–blue–green sequence can be obtained with (black, green) in slot #1, (red, blue) in #3, (blue, green) in #6, and (green, green) in #8. Generally, if all new orders get assigned to the same unit, a certain sequence can only be obtained with a single combination of tasks.

Overall, the final number of time slots will be larger for CT4I than for CT3I. Because of this, there may be idle slots between orders or empty last slots. Interestingly, for NOS = 1, one gets the exact same number of possible assignments (binary variables) per iteration for both models: 4, 10, 16, 22,.... For NOS = 2, iteration $j = 2$ requires 44 possible assignments for CT4I vs. 36 for CT3I, and the difference will grow with both NOS and j.

Mathematical Formulations

The formal mathematical formulations are given next, together with a brief review of the most relevant features. The reader is directed to previous articles[9,10,26] for further details. The most significant change is perhaps the introduction of additional dynamic sets to restrict the domain of variables and constraints, whose elements will change between iterations of the scheduling algorithm. For this reason, their exact definition is left to the following section.

CT3I

The unit-specific continuous-time formulation of Castro et al.[9] uses three-index binary variables $N_{i,m,t}$ to identify the execution of order i in unit m during time slot t (starting at event point t). Positive continuous variables $R_{m,t}$ keep track of equipment availability, timing variables $T_{t,m}$ give the absolute time of event point t in time grid m, while the transfer time to the next stage of order i after stage k is $\mathrm{TD}_{i,k}$. The remaining variables are the only ones involved in the objective function. MS is the makespan, while $S^1_{t,m}$ and S^2_i are slack variables that allow for the violation of due date constraints, penalized through weights α and β.

$$\min \mathrm{MS} + \sum_{m \in M_{|\mathrm{K}|}} \sum_{t \in T^{\mathrm{active}}_m} \alpha \cdot S^1_{t,m} + \sum_{i \in I^{\mathrm{active}}} \beta \cdot S^2_i. \tag{5}$$

Equation 6 ensures that at most one order is allocated to unit m during slot t. To reduce solution degeneracy, slots are filled from right to left as previously explained, recall Figure 4. This is the same as saying that a given unit starts idle (e.g., $R_{m,t-1} = 1$) and ends in processing mode (e.g., $R_{m,t} = 0$).

$$R_{m,t} = 1 - \sum_{i \in I_{m,t}} N_{i,m,t} \ \forall m \in M, t \in T^{\mathrm{act}}_m \tag{6}$$

$$R_{m,t} \leq R_{m,t-1} \ \forall m \in M, t \in T^{\mathrm{deg}}_m. \tag{7}$$

Equation 8 is the core of model CT3I and ensures that the difference in time between two consecutive event points (of a certain time grid) must be greater than the duration of the order being executed plus the required changeover time for the following order. This is illustrated in Figure 3.

$$T_{t+1,m}\big|_{t \notin T^{\mathrm{last}}_m} + \mathrm{MS}\big|_{t \in T^{\mathrm{last}}_m} - T_{t,m} \geq \sum_{i' \in I_{m,t}} N_{i',m,t} p_{i',m} + \left(N_{i,m,t} \mathrm{cl}^{\max}_{i,m} - \sum_{\substack{i' \in I_{m,t+1} \\ i' \neq i}} N_{i',m,t+1} \mathrm{cl}^{\Delta}_{i,i',m} \right)\Bigg|_{t \notin T^{\mathrm{last}}_m} \quad \forall m \in M, t \in T^{\mathrm{act}}_m, i \in I_{m,t}. \tag{8}$$

Equations 9 and 10 are the release and due date constraints, respectively. In cases where the due dates cannot be met, Eq. 10 is relaxed through slack variable $S^1_{t,m}$. Because of the form of the constraint, the slacks are needed for all pairs (slot, unit) and not just for those belonging to last stage units, which are penalized in the objective function (see Eq. 5).

$$T_{t,m} \geq \sum_{i \in I_{m,t}} N_{i,m,t} \times (r_i + \sum_{\substack{k' \in K \\ k' < k}} \min_{\substack{m' \in M_{k'} \\ m' \in M_i}} p_{i,m'}) \, \forall k \in K, m \in M_k, t \in T_m^{\text{act}} \tag{9}$$

$$T_{t,m} \leq \sum_{i \in I_{m,t}} N_{i,m,t} \cdot (d_i - p_{i,m} \cdot \sum_{\substack{k' \in K \\ k' > k}} \min_{\substack{m' \in M_{k'} \\ m' \in M_i}} p_{i,m'}) + S^1_{t,m} + H(1 - \sum_{i \in I_{m,t}} N_{i,m,t}) \, \forall k \in K, m \in M_k, t \in T_m^{\text{act}}. \tag{10}$$

The transfer time of order i in stage k must be greater than its finishing time in stage k and lower than its starting time in stage $k+1$.

$$\text{TD}_{i,k} \geq T_{t,m} + N_{i,m,t} p_{i,m} - H(1 - \sum_{\substack{t' \in T_m^{act} \\ t' \geq t \\ i \in I_{m,t'}}} N_{i,m,t'}) \quad \forall k \in K, k \neq |K|, m \in M_k, t \in T_m^{\text{act}}, i \in I_{m,t} \tag{11}$$

$$\text{TD}_{i,k-1} \leq T_{t,m} + H(1 - \sum_{\substack{t' \in T_m^{\text{act}} \\ t' \leq t \\ i \in I_{m,t'}}} N_{i,m,t'}) \quad \forall k \in K, k \neq 1, m \in M_k, t \in T_m^{\text{act}}, i \in I_{m,t}. \tag{12}$$

The transfer times for orders not involved in stage k equal those in the previous stage.

$$\text{TD}_{i,k} = \text{TD}_{i,k-1} \; \forall k \in K, i \in I^{\text{act}}, i \notin I_k. \tag{13}$$

All orders need to be executed once on every stage they need to go through.

$$\sum_{m \in M_k} \sum_{\substack{t \in T_m^{\text{act}} \\ i \in I_{m,t}}} N_{i,m,t} = 1 \; \forall k \in K, i \in I^{\text{act}} \cap I_k. \tag{14}$$

Although not required, previous studies[9] have found the following constraints to significantly improve the performance of the full-space model. Equations 15 and 16 act as lower and upper bounds on the transfer times, respectively. Notice the use of slack variable S_i^2, which allows the due date of order i to be violated. As variables $\text{TD}_{i,k}$ have considerable more freedom than their $T_{t,m}$ counterparts, it is more likely for slacks $S^1_{t,m}$ to be active so one should set $\alpha \gg \beta$. Equations 17 and 18 are responsible for reducing the integrality gap.

$$\text{TD}_{i,k} \geq r_i + \sum_{\substack{k' \in K \\ k' \leq k}} \sum_{m \in M_{k'}} \sum_{\substack{t \in T_m^{\text{act}} \\ i \in I_{m,t}}} N_{i,m,t} p_{i,m} \; \forall i \in I^{\text{act}}, k \in K, k \neq |K| \tag{15}$$

$$\text{TD}_{i,k} \leq d_i + S_i^2 - \sum_{\substack{k' \in K \\ k' > k}} \sum_{m \in M_{k'}} \sum_{\substack{t \in T_m^{\text{act}} \\ i \in I_{m,t}}} N_{i,m,t} p_{i,m} \quad \forall i \in I^{\text{act}}, k \in K, k \neq |K| \tag{16}$$

$$T_{t,m} \leq \text{MS} - \sum_{i \in I_{m,t}} N_{i,m,t} \times (p_{i,m} + \sum_{\substack{k' \in k \\ k' > k \\ i \in I'_k}} \min_{\substack{m' \in M_{k'} \\ m' \in M_i}} p_{i,m'}) + \sum_{\substack{t' \in T \\ t' \geq t}} \sum_{\substack{i \in I_{m,t'} \\ k = |K|}} [N_{i,m,t'} \times (p_{i,m} + \text{cl}^{\min}_{i,m}\Big|_{t' \notin T_m^{\text{last}}})] \forall k \in K, m \in M_k, t \in T_m^{\text{act}} \tag{17}$$

$$\text{TD}_{i,k} \leq \text{MS} - \sum_{\substack{k' \in K \\ k' > k}} \sum_{m \in M_{k'}} \sum_{\substack{T_m^{\text{act}} \\ i \in I_{m,t}}} N_{i,m,t} p_{i,m} \; \forall i \in I^{\text{act}}, k \in K, k \neq |K|. \tag{18}$$

CT4I

Formulation CT4I features four-index binary variables $\overline{N}_{i,i',m,t}$ to identify the execution of order i in unit m at time slot t followed by the required changeover time for order i' to immediately follow. When compared with CT3I, the additional index facilitates the writing of the timing constraints linking two consecutive event points and makes other constraints tighter because of the consideration of the actual changeover time, instead of the minimum changeover time (compare Eq. 30 to Eq. 17). On the other hand, an extra summation is involved in the terms of most of the constraints, but, more importantly, excess resource variables for units need to be disaggregated into the possible equipment states. As an example, positive continuous variable $C_{i,m,t} = 1$ indicates that unit m is idle at slot t at a state that enables it to process order i. Variables $C^0_{i,m}$ are used to determine the initial equipment state.

The excess resource balances over the equipment states are given by Eq. 19. Notice in the third term on the right-hand side that combined tasks with a single-order index (i,i,m) do not need to produce an equipment state as the selection of such a task assumes that no other order follows. Unfortunately, we can no longer explore its full potential by making $C_{i,m,t} = 0$ $(\forall i,m,t)$, like in the full-space model,[10] as the decomposition approach postulates one or more time slots between already assigned orders, which can remain idle (see Figure 7). In this respect, we have also tested the CT3I approach with CT4I, which by using as few slots as possible per iteration makes the use of such constraint possible. Regrettably, a worse computational performance was observed, and so the results are not shown in the article.

$$C_{i,m,t} = C^0_{i,m}\Big|_{t=1} + C_{i,m,t-1}\Big|_{t \neq 1} + \sum_{\substack{i' \in I_{i,m,t-1} \\ i' \neq i}} \overline{N}_{i',i,m,t-1} - \sum_{\substack{i' \in I_m \\ i \in I_{i'},m,t}} \overline{N}_{i,i',m,t} \; \forall i \in I^{\text{act}}, m \in M_i, t \in T_m^{\text{act}}. \tag{19}$$

Equation 20 limits the initial states of units to a single order.

$$\sum_{i \in I_m} C^0_{i,m} \leq 1 \; \forall m \in M. \tag{20}$$

The central timing constraint is given in Eq. 21. It features only unit and slot indices in the constraint domain making it tighter than Eq. 8 and compensating the generation of larger mathematical problems when compared with CT3I.

$$T_{t+1,m}\big|_{t\notin T_m^{\text{last}}}+\text{MS}\big|_{t\in T_m^{\text{last}}}-T_{t,m}\geq\sum_{i'\in I_m}\sum_{i\in I_{i',m,t}}\overline{N}_{i,i',m,t}\times(p_{i,m}+\text{cl}_{i,i',m})\forall m\in M,t\in T_m^{\text{act}}. \quad (21)$$

Equations 22 and 23 are the release and due date constraints, respectively, where the upper bound on the starting time of combined task (i,i',m), $ub_{i,i'm}$, is calculated through Eq. 24.

$$T_{t,m}\geq\sum_{i'\in I_m}\sum_{i\in I_{i'm,t}}\overline{N}_{i,i',m,t}\times(r_i+\sum_{\substack{k'\in K\\k'<k}}\min_{\substack{m'\in M_{k'}\\m'\in M_i}}p_{i,m'})\quad\forall k\in K,m\in M_k,t\in T_m^{\text{act}} \quad (22)$$

$$T_{t,m}\leq\sum_{i'\in I_m}\sum_{i\in I_{i'm,t}}\overline{N}_{i,i',m,t}\text{ub}_{i,i',m}+S_{t,m}^1+H(1-\sum_{i'\in I_m}\sum_{i\in I_{i',m,t}}\overline{N}_{i,i'm,t})\ \forall m\in M,t\in T_m^{\text{act}} \quad (23)$$

$$\text{ub}_{i,i',m}=\min(d_i-\sum_{k\in K_m}\sum_{\substack{k'\in K\\k'>k}}\min_{\substack{m'\in M_{k'}\\m'\in M_i}}p_{i,m'}-p_{i,m},d_{i'}-\sum_{k\in K_m}\sum_{\substack{k'\in K\\k'>k}}\min_{\substack{m'\in M_{k'}\\m'\in M_{i'}}}p_{i',m'}-p_{i,m}-\text{cl}_{i,i',m}-p_{i',m}\big|_{i\neq i'})\quad\forall i,i'\in I,m\in M_i\cap M_{i'}. \quad (24)$$

The two sets of timing variables $T_{t,m}$ and $\text{TD}_{i,k}$ are related through Eqs. 25 and 26. Note that processing in stage k can start immediately after the end of the processing part of the combined task in stage $k-1$ as can be seen in Figure 6.

$$\text{TD}_{i,k}\geq T_{t,m}+\sum_{\substack{i'\in I_m\\i\in I_{i',m,t}}}\overline{N}_{i,i',m,t}p_{i,m}-H(1-\sum_{\substack{t'\in T_m^{\text{active}}\\t'\geq t}}\sum_{\substack{i'\in I_m\\i\in I_{i',m,t'}}}\overline{N}_{i,i',m,t'})\ \forall k\in K,k\neq|K|,m\in M_k,t\in T_m^{\text{act}},i\in I_{m,t} \quad (25)$$

$$\text{TD}_{i,k-1}\leq T_{t,m}+H(1-\sum_{\substack{t'\in T_m^{\text{active}}\\t'\leq t}}\sum_{\substack{i'\in I_m\\i'\in I_{i',m,t'}}}\overline{N}_{i,i',m,t'})\quad\forall k\in K,k\neq 1,m\in M_k,t\in T_m^{\text{act}},i\in I_{m,t}. \quad (26)$$

Equation 27, together with Eqs. 5 and 13 (shared with CT3I), complete the set of mandatory constraints.

$$\sum_{m\in M_k}\sum_{t\in T_m^{\text{act}}}\sum_{\substack{i'\in I_m\\i\in I_{i',m,t}}}\overline{N}_{i,i',m,t}=1\ \forall k\in K,i\in I^{\text{act}}\cap I_k. \quad (27)$$

The performance enhancement constraints are given in Eqs. 28–31. Note in the third term on the RHS of Eq. 30 that all processing and changeover times of tasks executed at or after slot t are accounted for, when linking the timing variables of last-stage units to the makespan.

$$\text{TD}_{i,k}\geq r_i+\sum_{\substack{k'\in K\\k'\leq k}}\sum_{m\in M_{k'}}\sum_{t\in T_m^{\text{act}}}\sum_{\substack{i'\in I_m\\i\in I_{i',m,t}}}\overline{N}_{i,i',m,t}p_{i,m}\ \forall i\in I^{\text{act}},k\in K,k\neq|K| \quad (28)$$

$$\text{TD}_{i,k}\leq d_i+S_i^2-\sum_{\substack{k'\in K\\k'>k}}\sum_{m\in M_{k'}}\sum_{t\in T_m^{\text{act}}}\sum_{\substack{i'\in I_m\\i\in I_{i',m,t}}}\overline{N}_{i,i',m,t}p_{i,m}\quad\forall i\in I^{\text{act}},k\in K,k\neq|K| \quad (29)$$

$$T_{t,m}\leq\text{MS}-\sum_{i'\in I_m}\sum_{\substack{i\in I_{i'm,t}\\k\neq|K|}}\overline{N}_{i,i',m,t}\times(p_{i,m}+\sum_{\substack{k'\in K\\k'>k\\i\in I_{k'}}}\min_{\substack{m'\in M_{k'}\\m'\in M_i}}p_{i,m'})+\sum_{\substack{t'\in T\\t'\geq t}}\sum_{i'\in I_m}\sum_{\substack{i\in I_{i',m,t'}\\k=|K|}}[\overline{N}_{i,i',m,t'}\times(p_{i,m}+\text{cl}_{i,i',m})]\quad\forall k\in K,m\in M_k,t\in T_m^{\text{act}} \quad (30)$$

$$\text{TD}_{i,k}\leq\text{MS}-\sum_{\substack{k'\in K\\k'>k}}\sum_{m\in M_{k'}}\sum_{T_m^{act}}\sum_{\substack{i'\in I_m\\i\in I_{i',m,t}}}\overline{N}_{i,i',m,t}p_{i,m}\ \forall i\in I^{\text{act}},k\in K,k\neq|K|. \quad (31)$$

Sequencing variables model

One can use the sequencing concept instead of time grids to schedule multistage plants. In particular, models relying on general precedence SVs tend to be more efficient than their immediate precedence counterparts. The drawback is that they are not entirely rigorous as a certain combination of data may lead to the elimination of the global optimal solution from the feasible space.[9] Nevertheless, this will not be observed in the large majority of problems. The advantages of SVs over time grid–based models are as follows: (i) they need to be solved only once as there is no need to iterate over the number of time slots to prove optimality; (ii) they can be much faster at finding good solutions and so are preferable[25] for rescheduling purposes where large amounts of computational time are seldom available.

We rely on the continuous-time model of Harjunkoski and Grossmann[26] despite the addition of the sequence-dependent changeover terms and the nomenclature change. Binary variables $X_{i,i',k}$ (with $i'>i$) indicate if order i precedes order i' in stage k, while binaries $Y_{i,m}$ assign the execution of order i to unit m. Timing variables $Tf_{i,k}$ give the order's ending time in stage k. The difference between the ending times of any two orders at a particular stage can be determined through big-M constraints (Eqs. 32 and 33). Notice that the SVs are only relevant if both i and i' are assigned to the same unit.

$$Tf_{i',k}\geq Tf_{i,k}+p_{i',m}+X_{i,i',k}\text{cl}_{i,i',m}-\max_{i''\in I^{\text{act}}}d_{i''}\times(3-X_{i,i',k}-Y_{i,m}-Y_{i',m})\quad\forall i,i'\in I^{\text{act}},i'>i,k\in K,m\in M_k\cap M_i\cap M_{i'} \quad (32)$$

$$Tf_{i,k}\geq Tf_{i',k}+p_{i,m}+\text{cl}_{i',i,m}\times(1-X_{i,i',k})-\max_{i''\in I^{\text{act}}}d_{i''}\times(2+X_{i,i',k}-Y_{i,m}-Y_{i',m})\quad\forall i,i'\in I^{\text{act}},i'>i,k\in K,m\in M_k\cap M_i\cap M_{i'}. \quad (33)$$

Equation 34 is used to tighten the formulation, with the makespan being greater than the sum of the duration of all tasks executed on a particular unit plus the minimum over all active orders of the minimum duration in units belonging to subsequent (second term on the RHS) and previous stages (third term on the RHS), the latter being adjusted by the

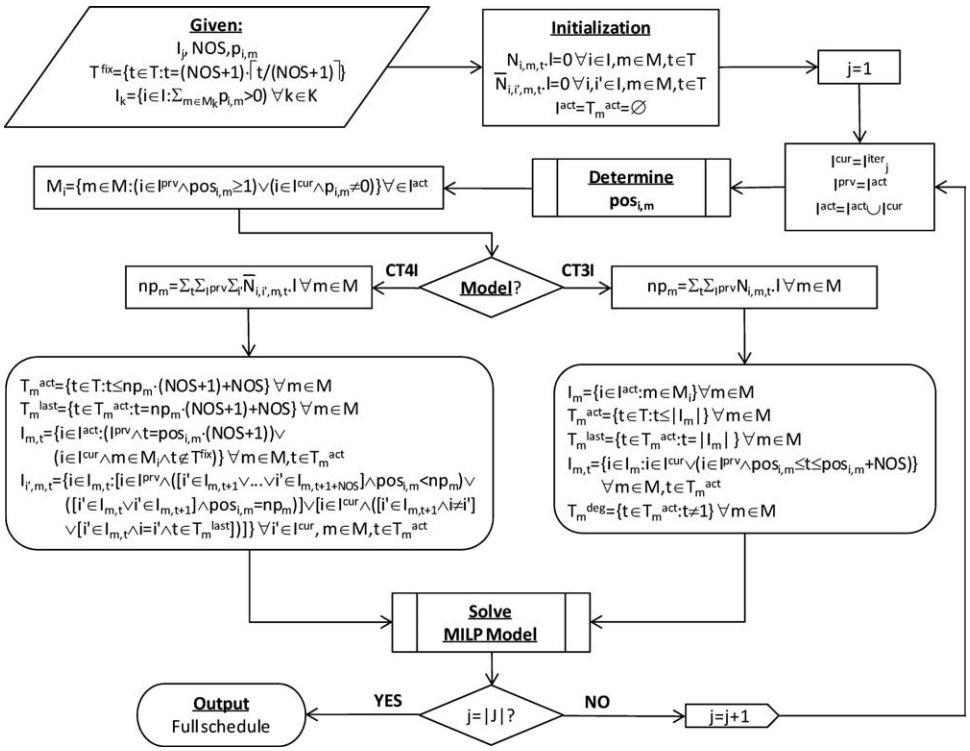

Figure 9. Constructive scheduling algorithm for time grid–based formulations.

release date. It is important to highlight that contrary to Eqs. 17 and 30 no changeover times are present, clearly showing that the concept of immediate precedence, implicit in time grid–based models, allows such type of constraints to be considerably tighter. The remaining sets of constraints are independent on whether the problem features sequence-dependent changeovers or not and can be found in Castro et al.[25]

$$\sum_{\substack{i\in I^{\mathrm{act}}\\ m\in M_i}} Y_{i,m}p_{i,m} \le \mathrm{MS} - \min_{\substack{i\in I^{\mathrm{act}}\\ m\in M_i}}\Big(\sum_{\substack{k'\in K\\ k'>k\\ i\in I_{k'}}} \min_{m'\in M_{k'}} p_{i,m'}\Big)$$

$$- \min_{\substack{i\in I^{\mathrm{act}}\\ m\in M_i}}\Big(r_i + \sum_{\substack{k'\in K\\ k'<k\\ i\in I_{k'}}} \min_{m'\in M_{k'}} p_{i,m'}\Big)\ \forall k\in K, m\in M_k, \bigcup_{i\in I^{act}} M_i \neq \varnothing\,. \quad (34)$$

Scheduling Algorithm

The primary goal of the scheduling algorithm is providing good solutions with few computational resources. It is greedy in the sense that it selects the optimal solution for each subproblem with the hope of finding a global optimum at the end of the last iteration. There is however no way of knowing if the resulting solution is indeed optimum other than solving the problem with a rigorous full-space formulation. Furthermore, we will not know how far the solution will be from the optimum as there is no obvious method of evaluating a tight lower bound on the value of the objective function, like relaxing integrality in an MILP model. Overall, there are a few parameters that influence the optimization runs and consequently the final outcome. These can be finetuned to ensure the best solution-quality/computational-effort tradeoff.

The proposed algorithm comprises two parts.[25] In the first, constructive scheduling, the goal is to find a good initial schedule by tackling the full set of orders, one or a couple of orders at a time, as previously explained. Then, we perform a local search to improve the solution. In this rescheduling step, one or a couple of orders are released from the schedule to try to find better unit assignments or sequences. It can be viewed as repeating the last iteration of the constructive step several times for different order candidates. The many tests conducted have shown that better solutions from the constructive step usually lead to better solutions at the end, and that most of the iterations of the rescheduling step are unsuccessful. For this reason, considerable more resources are allocated to the former step.

Constructive scheduling

The outcome of the constructive part of the algorithm is primarily influenced by three user decisions: (i) extent of the decomposition process, i.e., number of iterations, controlled by parameter NOS giving the number of orders per iteration that are scheduled with the full degree of freedom; (ii) distribution of orders over iterations, following a preordering heuristic, or a random choice and given by set I_j^{iter}; (iii) underlying mathematical formulation, either a time grid– or sequence-based approach. Other settings that can affect the output schedule and that are not explicit in Figure 9 are those from the MILP solver. Examples are the optimality tolerance and maximum resource limit. We have also observed that different versions of the solver generated solutions with a different value of the objective function, even in cases where all iterations were solved to zero optimality. It can be explained by the high degree of degeneracy of the

partial schedules and by the many iterations involved. Note that a different choice at an early iteration may steer the search into diverse regions of the solution space.

The mathematical formulations have been presented in a general form, with the domain of variables and constraints defined using dynamic sets. The algorithm will simply change the elements of such sets according to the value of some model variables from the preceding iteration and problem data. To distinguish the variable from its value, suffix (.*l*) is used for the latter.

Every iteration j starts with the selection of orders that: (a) are being considered for the first time, I^{cur}; (b) have previously been considered, I^{prv}; (c) are under consideration, I^{act}. The next step in Figure 9 is to determine the position in the sequence of previously scheduled orders. Parameter $\text{pos}_{i,m}$ can be easily calculated based on the value of the binary variables (see Castro et al.[25] for CT3I). If positive, unit m can handle order i ($m \in M_i$). Current orders with nonzero processing times are also elements of the set. The remainder of the algorithm depends on the mathematical formulation being used. Figure 9 provides all the necessary information for CT3I and CT4I, while the code for SV can be found in Castro et al.[25]

For CT4I, on the left, the number of active time slots for unit m, T_m^{act}, depends on the number of previously assigned orders, np_m, and on NOS. As illustrated in Figures 7 and 8, any previous order can be assigned to a single fixed slot $t \in T^{\text{fix}}$, determined by its position in the sequence (see definition of set $I_{m,t}$, which gives the orders that in unit m can be executed in slot t) for a total of np_m slots. Current orders can be assigned to any other slot, either before previous orders or after the last one for a total of $\text{np}_m \times \text{NOS} + \text{NOS}$ slots. The last slot is the sole element of set T_m^{last}. Finally, set $I_{i',m,t}$ gives the orders that can be assigned to slot t of unit m and be followed by order i'. It requires a significant number of conditions that achieve similar outputs to those illustrated in the earlier described Figures 7 and 8.

For CT3I it is more straightforward to update the elements of the dynamic sets. The number of active slots is equal to the number of orders that have been or can be allocated to the unit. Current orders can then be assigned to any slot, while previously assigned orders can be assigned to just $\text{NOS} + 1$ slots, from $\text{pos}_{i,m}$ forward, as can be seen in Figures 4 and 5. Finally, the solution degeneracy reduction constraint (Eq. 7) is to be written for all active slots but the first.

Once all sets have been updated, the optimal schedule for the active set of orders is found through the solution of the corresponding MILP model. We then proceed to the next iteration until all orders have been scheduled. At that point, the constructive scheduling algorithm terminates by reporting the final solution, which will be the starting point for the rescheduling algorithm.

Rescheduling

Although solutions resulting from the constructive scheduling algorithm can be considered suitable for most practical purposes, there are cases where we can do much better by performing rescheduling with few additional computational resources. The most relevant situation occurs when the returned solution is unable to meet all due date constraints merely because of the decomposition process. Previous test studies on similar problems without sequence-dependent changeovers have shown a successful elimination of all due date violations, following a local search around the best schedule. The reader is once more directed to Castro et al.[25] for details concerning the rescheduling algorithm, which is based on the original ideas of Roslöf et al.[18] and Méndez and Cerdá.[19] It relies on the general precedence SVs model and not on any of the alternative time grid formulations, essentially because the former is faster at finding the optimal solution. Note that multiple trials and limited computational resources per iteration are involved.

Besides the underlying model, the rescheduling algorithm was run with the exact same primary settings NOS and I_j^{iter} of the constructive part. Recall that increasing the NOS value can potentially lead to better solutions. Keeping I_j^{iter} unchanged means starting with those orders that were scheduled first, whose unit assignments were chosen without any knowledge of their competitors processing data and thus have a higher improvement potential. We have nevertheless tried to change the order-iteration assignments using other heuristics but did not observe consistently better results, and so they are not reported in the article.

Computational Results

The performance of the scheduling algorithm is now illustrated through the solution of 10 example problems. Problems P7–P13 have been addressed before[9,10] and can be solved to global optimality by the full-space continuous-time scheduling formulations, considered in this article, as well as by a constraint programing model. Their main purpose is to evaluate the quality of the solution returned by the algorithm. The challenging problem is P16, which represents a pharmaceutical batch plant and consists of 50 orders, 17 units, and six stages. It does neither involve release nor due dates. For the former problem set, we have made $\alpha = 10$ and $\beta = 1$ (Eq. 5) to prioritize schedules that meet the due date constraints. Problems P14 and P15 consider the first 30 and 40 orders of P16 to measure and illustrate the effect of problem size on computational effort.

The algorithm and underlying models were implemented in the GAMS 23.2 with CPLEX 12.1 (default options) as the MILP solver. The termination criteria were a relative optimality tolerance equal to 10^{-6} and a maximum computational time equal to: (i) 3600 CPUs per iteration on the constructive part; (ii) 60 CPUs per iteration on the rescheduling part. The hardware consisted of a laptop with an Intel Core2 Duo T9300 2.5 GHz processor, with 4 GB of RAM running Windows Vista Enterprise.

Full-space models

We start by analyzing the performance of the full-space models. As can be seen in Table 1, time grid model CT3I emerges as the best performer followed by CT4I and SV, the latter being unable to find the optimal solution for P13 up to the maximum resource limit. It is important to highlight the solver and hardware developments that have occurred in roughly 3 years, which have led to orders of

Table 1. Computational Performance of Full-Space Models (CPUs)

Problem	Features ($\|I\|$,$\|M\|$,$\|K\|$)	Optimum/Best	Time Slots ($\|T\|$)	CT3I	CT4I	SV
P7	(8,6,2)	542	3	4.84	5.37	4.81
P8	(8,6,2)	584	3	1.54	0.68	27.0
P9	(8,6,3)	915	4	10.5	20.0	19.7
P10	(8,6,3)	914	4	1.26	5.25	147
P11	(12,6,2)	233	5	227	1816	153
P12	(8,8,4)	265	4	30.6	73.1	50.5
P13	(15,4,2)	273	8	7871	25142	24,105*,†
P14	(30,17,6)	49.973	–	–	–	6.38
		43.052	–	–	–	600
		36.121	–	–	–	1200
		36.121	–	–	–	51,900*
P15	(40,17,6)	79.928	–	–	–	21.5
		67.475	–	–	–	600
		67.475	–	–	–	3600
		55.220	–	–	–	60,000†
P16	(50,17,6)	79.995	–	–	–	78.3
		78.125	–	–	–	600
		78.003	–	–	–	3600
		64.583	–	–	–	60,000‡

*Out of memory termination.
†Suboptimal solution = 275, best possible = 259.
‡Maximum resource limit.

magnitude savings in computational time. For instance, SV ran out of memory at 23,408 CPUs with CPLEX 9.1,[9] while now P11 can be solved in just 153 CPUs. For CT3I/4I the results are for time grids with the number of slots listed in column 4, the minimum values for which the optimal solution can be found. Note that the unit-specific grids use the same number of slots, whereas in the decomposition algorithm they will typically grow at different rates.

The solution dependency on the number of slots, which can only be roughly estimated a priori, makes SV a more suitable candidate for large-scale problems. One may always expect a feasible outcome, provided that the time horizon is sufficiently large. Interestingly, the first feasible solution is roughly equal to the specified value (50 for P14 and 80 for P15 and P16). The results in Table 1 show that such solutions improve significantly given more computational resources. The best makespans from the full-space SVs model were equal to 36.121, 55.220, and 64.583. For P14, the solution was found in half an hour with no improvement in the next 14 h; for P15 the absolute decrease in makespan in the first hour was identical to that observed afterward, whereas for P16 the major improvements occur after the hour. Most importantly, such values are still distant (16, 30, and 35%) from the best ones found by the decomposition algorithm in considerably less time (see Table 2) degrading with the increase in problem size. This clearly highlights the value of the proposed decomposition algorithm.

Algorithm validation

The solutions obtained by the scheduling algorithm are listed in Table 2 for a total of 90 trials resulting from different choices in terms of number of orders scheduled at a time and underlying model in the constructive part. The smaller instances (P7–P13) can be used for validating the effectiveness of the scheduling algorithm. As all generated subproblems were solved to optimality, failure to find the global optimal solution is entirely due to the decomposition strategy. Interestingly, there was a single successful run with respect to finding the global optimal solution (CT3I for P12 with NOS = 2) in the 63 tests, which leaves room for future improvements. Note however that the final result often changes between models (see Table 2) because of different choices of intermediate degenerate solutions that direct the algorithm to other parts of the feasible region, suggesting that the best option is perhaps using different settings in parallel and then selecting the best solution of the lot. It is beyond the scope of the article to further explore this aspect.

Table 2. Solution from Scheduling Algorithm for Different Model Choices and NOS Values

	NOS = 1			NOS = 2			NOS = 3		
Problem	CT3I	CT4I	SV	CT3I	CT4I	SV	CT3I	CT4I	SV
P7		591		582	595		567	**558**	569
P8	657	664	609	**584**	611	**584**	611	587	595
P9	*1355*	*1355*	*1374*		**981**			*1166*	
P10		970		**937**	938	**937**		938	
P11	270	261	254	248	**245**	249		261	
P12		*374*		**265**	*507*	*305*		275	
P13	314		**295**	**295**	328	306	307	309	304
P14	31.494	33.424	32.389	31.300	**31.018**	32.725	33.100	31.934	31.157
P15	42.298	42.056	43.989	43.159	40.766	43.400	42.698	42.151	**40.299**
P16	50.600	52.849	52.093	51.005	48.929	49.342	53.948	51.444	**47.722**

Best solution in bold, solution with violation of due dates in italic.

Table 3. Scheduling Algorithm Computational Time (CPUs) for Different Model Choices and NOS Values

	NOS = 1			NOS = 2			NOS = 3		
Problem	CT3I	CT4I	SV	CT3I	CT4I	SV	CT3I	CT4I	SV
P7	3.79	3.89	2.14	2.34	2.26	2.46	2.08	4.10	1.34
P8	3.97	3.78	2.14	2.25	2.38	1.14	2.13	2.27	1.10
P9	3.74	3.93	2.16	3.78	2.34	1.31	7.09	13.0	1.77
P10	4.01	3.79	2.10	5.20	3.22	1.26	2.75	4.91	1.34
P11	6.05	5.86	3.14	4.05	4.01	1.92	8.36	12.9	2.37
P12	4.27	4.12	2.18	10.7	7.42	1.96	26.8	27.2	1.82
P13	7.62	7.54	4.25	7.69	5.50	2.82	15.4	21.1	4.35
P14	235	40.9	47.7	26872	1607	10958	25664	20540	9468
P15	796	99.7	344.8	39047	3251	8580	34046	30409	19687
P16	2958	371	196.7	61940	12690	18571	52197	29413	30173

Although the model influence in solution quality can be considered stochastic, it is clearly better to consider as many orders as possible (higher NOS value) and thus, work as close as possible to full-space mode (NOS = $|I|$). In terms of relative optimality gap, the solutions listed in Table 2 lead to average values equal to 20.0, 10.8, and 9.2% for NOS = 1, 2, and 3, respectively. Note that such values are somewhat distorted by the 17% of schedules that exhibit violation of the due date constraints, which recall, are severely penalized in the objective function. This is a relatively high frequency when compared with outcomes with no due date violations obtained in problems of similar size but without changeovers.[25] In that study, the optimal solution was also found in 33% of the cases, clearly reflecting the added complexity of bringing sequence-dependent changeovers into play.

Model choice

We now focus on the large-scale problems P14–P16, which are the primary targets of the scheduling algorithm. If one of the goals is to obtain very good solutions, the other is obtaining them with small computational times. From the total efforts listed in Table 3, one can see that reasonably good solutions can be obtained up to roughly 6 min of computational time when using CT4I or SV with NOS = 1. If this is still too long, one could try to use a simpler, largely numerical or enumerative search strategy and evaluate its solution quality performance to see if it is a better alternative to solving an MILP at each iteration. Again, this is beyond the scope of the article.

The resulting models from CT3I are clearly more difficult to solve, which contradicts the behavior of the full-space version. The point to make is that the best conceptual alternative for a given scenario may no longer be appropriate in another, and one should have a set of competitive models rather than a single one. This has also been highlighted by Liu and Karimi,[27] when analyzing the results for 11 different formulations for this particular system. In this respect, the resulting solutions were now all different, despite solving to optimality all subproblems in the constructive part.

In terms of the comparison between CT4I and SV, there is no clear winner. On the one hand, CT4I is the best performer for NOS = 2 both in solution quality and total computational effort, being able to still solve P14 and P15 in less than 1 h. The solutions then degrade for NOS = 3, primarily because we can no longer solve all subproblems to optimality up to the 1-h maximum resource limit. This effect is not noticeable for SV, probably because of its better ability to find very good solutions in the early nodes of the search tree, even though it may be very hard to close the optimality gap down to zero. As a consequence, the best solutions for SV were found for NOS = 3, using computational times of the same order of magnitude as those for NOS = 2, as the increased difficulty per problem is somewhat compensated by the need to solve fewer of them (recall Eq. 1).

Remarks

The major strength of the proposed algorithm is its ability to address problems systematically for a wide variety of settings. The results have shown that we should look beyond the traditional measures of solution quality and total computational times. Nevertheless, finding a better solution than the best published is always a big motivation for researchers and therefore should not be neglected. We previously[28] found solutions with a makespan equal to 30.480 and 50.721 h for P14 and P16, respectively. More recently, Kopanos and Puigjaner[29] have shown a 29.91-h schedule in the poster session of ESCAPE-19. We did slightly better (29.871 h) with CT4I, NOS = 2, and a different ordering heuristic for rescheduling, before upgrading from CPLEX 11.1. However, we prefer to show in Figure 10 the best schedule for P16, the most challenging problem (50 orders), which has seen a larger improvement. With the data supplied as Supporting Information, the reader can possibly find better solutions.

The focus has been on mathematical programing techniques of the MILP type. They are easy to implement as one can use powerful modeling systems like GAMS and AIMMS, where there is a clear separation between model and solution method. Although evolutionary algorithms such as genetic algorithms and tabu search are possible alternatives for this problem, they require the user to write procedures to perform both the modeling and the solution method, which can be time consuming and more prone to errors. Furthermore, there is no explicit capability for handling constraints. Overall, MILP techniques are continually improving in terms of speed of computation.[30] This means that the capability of solving larger and more complex problems will only increase in the future, whereas for evolutionary algorithms the only major speed up that can be expected is due to increased speed in computers, which would of course also benefit MILP techniques. Evidence that deterministic

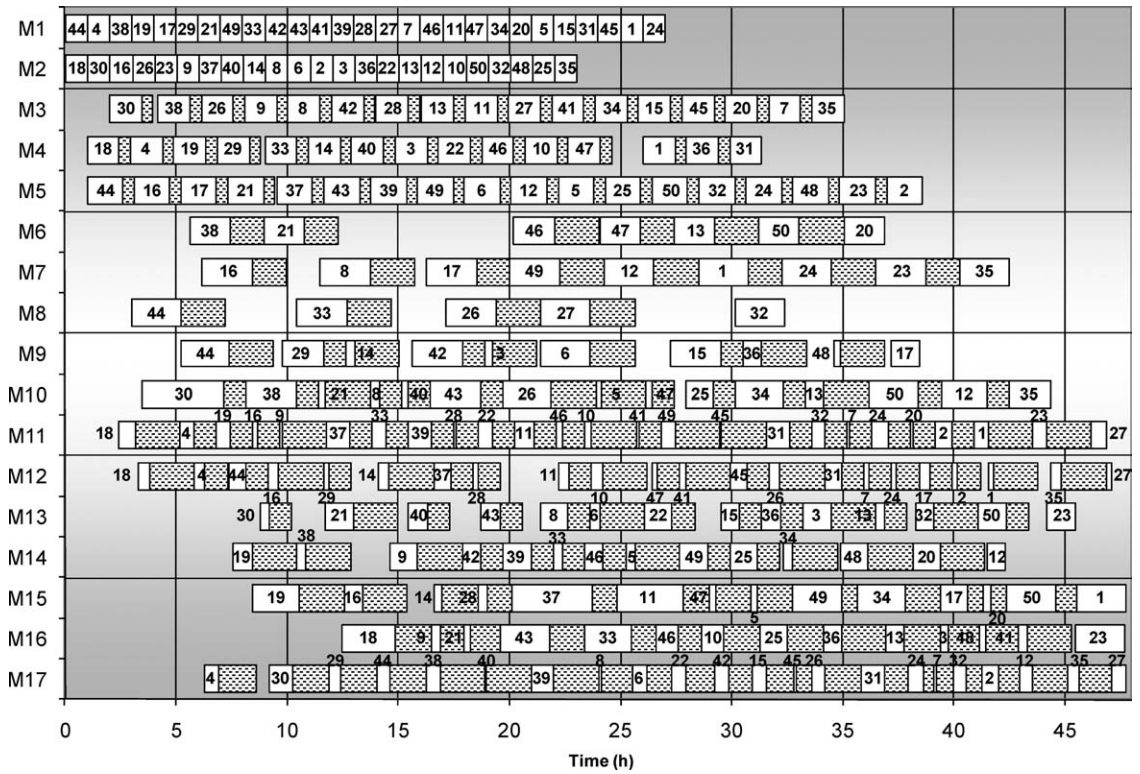

Figure 10. Best found solution for P16 (makespan = 47.722 h).

methods like MILP tend to outperform evolutionary algorithms can be seen in Ref. [31].

Conclusions

This article has addressed the scheduling of large-scale multistage batch plants with sequence-dependent changeovers. A new decomposition algorithm has been proposed to construct the full schedule in several iterations. It is divided into two parts. In the first, the full set of orders is handled sequentially according to some heuristic. Newly considered orders are given full degrees of freedom in terms of unit assignments and sequencing, while those handled in previous iterations are allowed to change their starting times provided that their unit assignments and relative positions in the sequence are kept fixed. After generating an initial schedule, a similar process is used in the second part of the algorithm. One, two, or three orders are picked per iteration with the aim of finding a better solution through rescheduling.

By changing the number of orders tackled per iteration, the algorithm can effectively handle problems of different size in an efficient way and hence exploit at a maximum the available computational resources. In the core of the algorithm are well-established continuous-time mixed-integer linear programing formulations, with three conceptually different options being available to the user. Although a three-index binary variable unit-specific formulation has been shown to be the best full-space performer, the corresponding decomposition strategy generates harder subproblems than their counterparts, which is translated into longer computational times and worse solutions. Better results have been achieved with a four-index unit-specific formulation and with a SVs model, which have complementary strengths. Whereas the latter typically finds the optimal solution faster, the former is superior at closing the integrality gap and hence at proving optimality.

Overall, the proposed algorithm has successfully tackled a real-life problem in a few minutes of computational time and thus has the potential to support the decision making for industrial-scale problems.

Notation

Sets

$I/i, i'$ = orders
I^{act} = active orders for scheduling model
I^{cur} = orders being currently considered
I_j^{iter} = orders being considered for the first time in iteration j
I^{prv} = previously scheduled orders
I_k = orders processed in stage k
I_m = orders that can be executed in unit m
$I_{m,t}$ = orders that can be processed in unit m at interval t
$J/j, j'$ = iterations of constructive scheduling algorithm
$K/k, k'$ = stages
$M/m, m'$ = units
M_i = units that can process order i
M_k = units belonging to stage k
$T/t, t'$ = time slots (event points)
T_m^{act} = active event points for unit m
T_m^{deg} = vacant time slots in unit m whose predecessor is also free
T^{fix} = event points with fixed orders
T_m^{last} = last active event point in unit m

Parameters

$\text{cl}_{i,m}^{\max}$ = maximum changeover time from order i in unit m
$\text{cl}_{i,i',m}^{\Delta}$ = difference between maximum and actual changeover from order i to i' in unit m
$\text{cl}_{i,i',m}$ = duration of changeover task from order i to i' in unit m

d_i = due date of order i
H = time horizon
NOS = number of orders to schedule with total freedom
np_m = number of orders already assigned to unit m
$p_{i,m}$ = processing time of order i in unit m
$pos_{i,m}$ = position of order i in the sequence of unit m
r_i = release date of order i
$span_i$ = processing time window for order i
$ub_{i,i',m}$ = highest time at which order i (followed by order i') can start to be processed in unit m
α = contribution of first type of slack variables to the objective function
β = contribution of second type of slack variables to the objective function

Variables

$C_{i,m,t}$ = excess amount of equipment state associated to order i of unit m at time point t
$C^0_{i,m}$ = initial amount of equipment state associated to order i of unit m
MS = makespan
$N_{i,m,t}$ = binary variable that identifies the execution of order i in unit m at interval t
$\overline{N}_{i,i',m,t}$ = binary variable that assigns the start of order i (followed by i') in unit m at time point t
$T_{t,m}$ = absolute time of event point t on unit m
$TD_{i,k}$ = transfer time of order i in stage k
$R_{m,t}$ = excess amount of unit m at event point t
$S^1_{t,m}$ = slack variable associated to event point t unit m
S^2_i = slack variable associated to order i
$Tf_{i,k}$ = ending time of order i in stage k
$X_{i,i',k}$ = binary variable indicating that order i globally precedes order i' in stage k
$Y_{i,m}$ = binary variable that identifies the execution of order i in unit m

Variable attribute

$.l$ = current value (level)

Literature Cited

1. Grossmann IE. Enterprise-wide optimization a new frontier in process systems engineering. *AIChE J*. 2005;51:1846–1857.
2. Varma VA, Reklaitis GV, Blau GE, Pekny JF. Enterprise-wide modeling and optimization—an overview of emerging research challenges and opportunities. *Comput Chem Eng*. 2007;31:692–711.
3. Harjunkoski I, Nyström R, Horch A. Integration of scheduling and control—theory of practice? *Comput Chem Eng*. 2009;33:1909–1918.
4. Floudas CA, Lin X. Continuous-time versus discrete-time approaches for scheduling of chemical processes: a review. *Comput Chem Eng*. 2004;28:2109–2129.
5. Méndez CA, Cerdá J, Grossmann IE, Harjunkoski I, Fahl M. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Comput Chem Eng*. 2006;30:913–946.
6. Wassick J. Enterprise-wide optimization in an integrated chemical complex. *Comput Chem Eng*. 2009;33:1950–1963.
7. Henning G. *Production scheduling in the process industries: current trends, emerging challenges and opportunities*. In: Alves R, Nascimento C, Biscaia E Jr, editors. *Computer Aided Chemical Engineering*, *Vol*. 27. Amsterdam: Elsevier, 2009:23.
8. Pantelides CC. *Unified frameworks for the optimal process planning and scheduling*. In: Rippin DWT, Hale JC, Davis J, editors. *Proceedings of the Second Conference on Foundations of Computer Aided Operations*. New York: Cache Publications, 1994:253.
9. Castro PM, Grossmann IE, Novais AQ. Two new continuous-time models for the scheduling of multistage batch plants with sequence dependent changeovers. *Ind Eng Chem Res*. 2006;45:6210–6226.
10. Castro PM, Novais AQ. Scheduling multistage batch plants with sequence-dependent changeovers. *AIChE J*. 2009;55:2122–2137.
11. Castro PM, Erdirik-Dogan M, Grossmann IE. Simultaneous batching and scheduling of single stage batch plants with parallel units. *AIChE J*. 2008;54:183–193.
12. Sundaramoorthy A, Maravelias CT. Simultaneous batching and scheduling in multistage multiproduct processes. *Ind Eng Chem Res*. 2008;47:1546–1555.
13. Erdirik-Dogan M, Grossmann IE. Slot-based formulation for the short-term scheduling of multistage batch plants with sequence-dependent changeovers. *Ind Eng Chem Res*. 2008;47:1159–1183.
14. Ku H-M, Karimi IE. Scheduling in serial multiproduct batch processes with finite intermediate storage: a mixed integer linear program formulation. *Ind Eng Chem Res*. 1998;27:1840–1848.
15. Dannenbring DG. An evaluation of flowshop sequencing heuristics. *Manag Sci*. 1977;23:1174.
16. Ku H-M, Karimi IE. Completion time algorithms for serial multiproduct batch processes with shared storage. *Comput Chem Eng*. 1990;14:49–69.
17. Ku H-M, Karimi IE. Scheduling algorithms for serial multiproduct batch processes with tardiness penalties. *Comput Chem Eng*. 1991;15:283–286.
18. Roslöf J, Harjunkoski I, Björkqvist J, Karlsson S, Westerlund T. An MILP-based reordering algorithm for complex industrial scheduling and rescheduling. *Comput Chem Eng*. 2001;25:821–828.
19. Méndez C, Cerdá J. Dynamic scheduling in multiproduct batch plants. *Comput Chem Eng*. 2003;27:1247–1259.
20. Dimitriadis AD, Shah N, Pantelides CC. RTN-based rolling horizon algorithms for medium term scheduling of multipurpose plants. *Comput Chem Eng*. 1997;21:S1061.
21. Lin X, Floudas CA, Modi S, Juhasz NM. Continuous-time optimization approach for medium-range production scheduling of a multiproduct batch plant. *Ind Eng Chem Res*. 2002;41:3884.
22. Janak SL, Floudas CA, Kallrath J, Vormbrock N. Production scheduling of a large-scale industrial batch plant. I. Short-term and medium-term scheduling. *Ind Eng Chem Res*. 2006;45:8234–8252.
23. Janak SL, Floudas CA, Kallrath J, Vormbrock N. Production scheduling of a large-scale industrial batch plant. II. Reactive scheduling. *Ind Eng Chem Res*. 2006;45:8253–8269.
24. Ierapetritou MG, Floudas CA. Effective continuous-time formulation for short-term scheduling. I. Multipurpose batch processes. *Ind Eng Chem Res*. 1998;37:4341–4359.
25. Castro PM, Harjunkoski I, Grossmann IE. Optimal short-term scheduling of large-scale multistage batch plants. *Ind Eng Chem Res*. 2009;48:11002–11016.
26. Harjunkoski I, Grossmann IE. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Comput Chem Eng*. 2002;26:1533–1552.
27. Liu Y, Karimi IA. Scheduling multistage, multiproduct batch plants with nonidentical parallel units and unlimited intermediate storage. *Chem Eng Sci*. 2007;62:1549–1566.
28. Castro P, Méndez C, Grossmann I, Harjunkoski I, Fahl M. *Efficient MILP-based solution strategies for large-scale industrial batch scheduling problems*. In: Marquardt W, Pantelides C, editors. *Computer Aided Chemical Engineering*, *Vol*. 21. Amsterdam: Elsevier, 2006:2231.
29. Kopanos G, Puigjaner L. *A MILP scheduling model for multi-stage batch plants*. In Jezowski J, Thullie J, editors. *Computer Aided Chemical Engineering*, *Vol*. 26. Amsterdam: Elsevier, 2009:369.
30. Bixby RE. Solving real world linear programs: a decade and more of progress. *Oper Res*. 2002;50:3–15.
31. Sahinidis N. Derivative-free optimization: algorithms, software and applications. Available at: http://egon.cheme.cmu.edu/ewocp/docs/SahinidisEWO_DFO2010.pdf. 2010.

Manuscript received Nov. 20, 2009, and revision received Mar. 12, 2010.