Greedy Algorithm for Scheduling Batch Plants with Sequence-Dependent Changeovers

Pedro M. Castro[*]

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

Iiro Harjunkoski

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

Ignacio E. Grossmann

Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

This paper presents a new algorithm for the near optimal scheduling of 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 of previously scheduled orders without altering the unit assignment and sequencing decisions, so that the combinatorial complexity is kept at a manageable level. The algorithm has been implemented for three alternative continuous-time mixed integer linear programming models and tested through the solution of ten

[*] To whom correspondence should be addressed. Tel.: +351-210924643. Fax: +351-217167016. E-mail: pedro.castro@ineti.pt

example problems for different decomposition settings. The results show that an industrial-size scheduling problem with 50-orders, 17-units distributed over 6-stages can effectively be solved in roughly 6 minutes of computational time.

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 (EWO). The optimal operation of manufacturing plants is naturally involved and includes different levels of decisions processes 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, cutting-stock problems, etc. 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 (PSE) 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 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 between products 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 programming formulations is limited to relatively small problems.

In order 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 been increasingly oriented towards 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 near optimal solutions rapidly, typically within a few minutes of computational time.

A very effective approach has been proposed by Roslöf et al[14] to solve a problem from a paper-converting mill with a single processing unit. Starting with an initial solution, rescheduling is achieved by releasing a subset of jobs and inserting them back into the schedule between the other fixed jobs. A key element is that every iteration contains information of the entire system meaning that the solution quality can either be improved or maintained. The same concept can be used to insert a new set of jobs and extended to plants with units in parallel[15], where rescheduling may involve re-sequencing as well as reallocation of jobs to units.

The movement from full-space to decomposition methods that work 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[16] approach is typically employed. 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 sub-horizon. The approach in Lin et al.[17] and Janak et al.[18] approach was 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[19].

The main drawback of decomposition approaches is that there is rarely an indication of 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 number of slots that guarantees optimality cannot accurately be predicted, which becomes a serious issue since a single increase in the number of slots typically results in a one order of magnitude increase in computational effort. Thus, a decomposition algorithm relying on a time grid 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.[18-19] when using the unit-specific model of Ierapetritou and Floudas[20] in their large-scale multipurpose plant study from BASF.

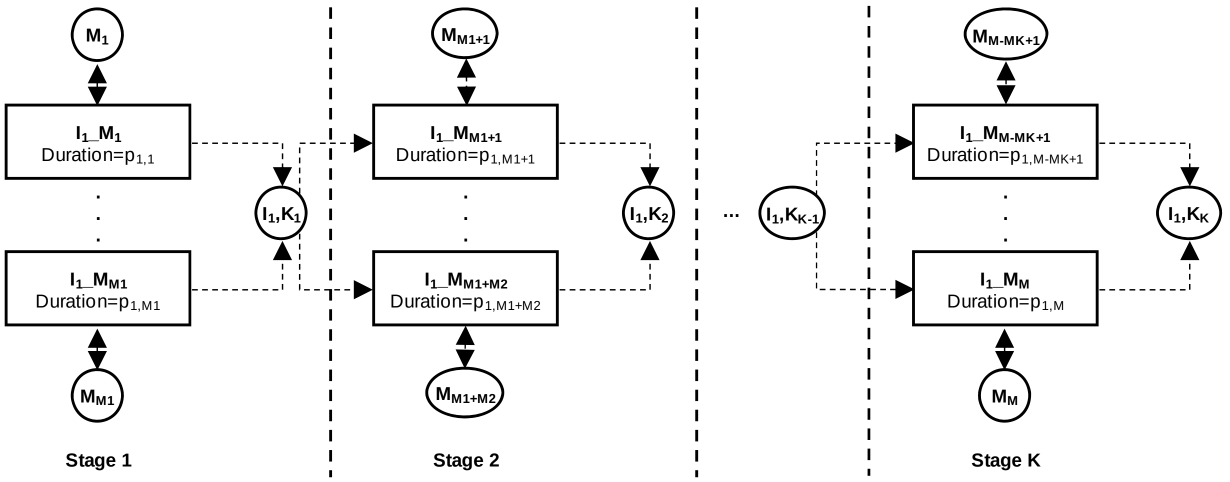
This article proposes a new algorithm for the short-term scheduling of multistage batch plants. It uses essentially the same ideas of Roslöf et al.[14] and Méndez and Cerdá[15], which have been recently[21] implemented with a unit-specific formulation, together with a sequence based continuous-time model. Due to the similar performance between the two approaches, we now consider the same plant layout, but with equipment units subject to sequence-dependent changeovers between orders. The non-trivial extension results from the existence of two alternative unit-specific models[9] that either consider changeovers implicitly, with 3-index binary variables (like previously), or explicitly, through 4-index binaries with combined processing and changeover tasks. Interestingly, it is the former choice that leads to major changes in the decomposition strategy. Most algorithmic features are kept, most importantly the fact that the determination of the number of time slots is unveiled as part of the search strategy without compromising tractability of the mathematical problem. Furthermore, time grids associated to different units will typically grow at different rates instead of using the same value in all grids, thus reducing solution degeneracy and problem size.

Besides the three alternative continuous-time formulations that can be employed, 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 to the corresponding full-space formulations for three distinct solution strategies. The method is applied to three industrial size problem instances, where the objective is to minimize the makespan. In a real scheduling environment minimizing the makespan alone does not always make sense due to 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

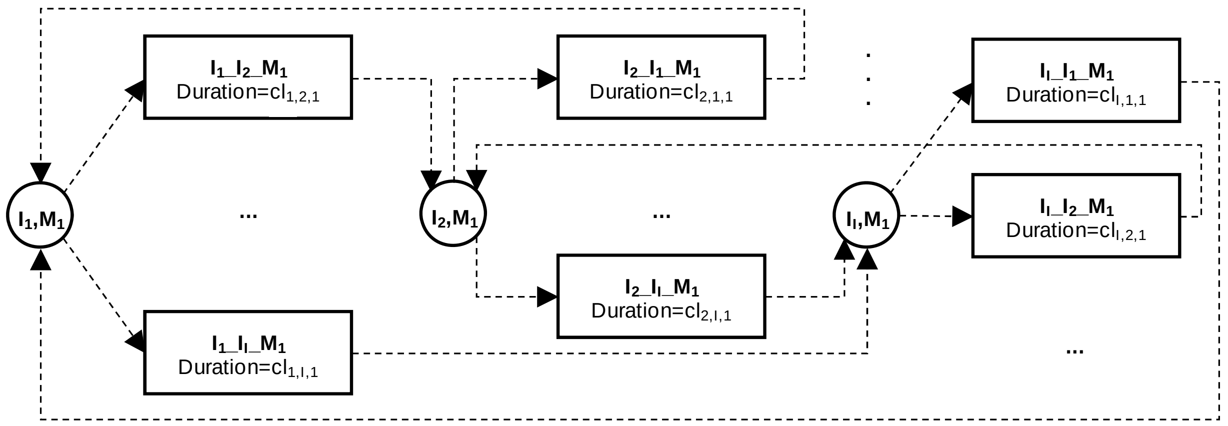
Given a multistage, multiproduct plant with $k \in K$ processing stages, $i \in I$ product orders and $m \in M$ units, the goal is to determine the assignment of orders to units and the sequence of orders at each unit so as to minimize the makespan. Orders may be subject to release (r_i) and due dates (d_i), which are enforced as hard constraints. The processing times are unit dependent, $p_{i,m}$, and the changeover times, $cl_{i,i',m}$, are sequence dependent. 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 (UIS/UW) 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 I_1, in the form of a Resource-Task Network[8]. The plant resources (circles) are the equipment units and material states, while 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.



M_1
$I_1\_M_1$ Duration=$p_{1,1}$
$I_1\_M_{M1}$ Duration=$p_{1,M1}$
M_{M1}
I_1,K_1
M_{M1+1}
$I_1\_M_{M1+1}$ Duration=$p_{1,M1+1}$
$I_1\_M_{M1+M2}$ Duration=$p_{1,M1+M2}$
M_{M1+M2}
I_1,K_2
I_1,K_{K-1}
M_{M-MK+1}
$I_1\_M_{M-MK+1}$ Duration=$p_{1,M-MK+1}$
$I_1\_M_M$ Duration=$p_{1,M}$
M_M
I_1,K_K
Stage 1
Stage 2
Stage K

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



$I_1\_I_2\_M_1$ Duration=$cl_{1,2,1}$
$I_2\_I_1\_M_1$ Duration=$cl_{2,1,1}$
$I_I\_I_1\_M_1$ Duration=$cl_{I,1,1}$
I_1,M_1
I_2,M_1
I_I,M_1
$I_I\_I_2\_M_1$ Duration=$cl_{I,2,1}$
$I_2\_I_I\_M_1$ Duration=$cl_{2,I,1}$
$I_1\_I_I\_M_1$ Duration=$cl_{1,I,1}$

Figure 2. The possible cleaning tasks for a certain unit (M_1) increase with the number of orders.

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 M_1, the possible states are illustrated in Figure 2. Generally speaking, cleaning tasks (i,i',m) changes 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. Due to 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 paper 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; (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. While 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, see Eq. 1. 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. To select the sequence in which the orders are inserted into the schedule (set I_j 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 (Eq. 2) are thus scheduled first. We have chosen the MST heuristic since it performed better than the earliest due date heuristic in previous work[21]. Other heuristics can naturally be used, which will probably lead to significant differences in both solution quality and computational effort.

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

$$span_i^{MST} = d_i - r_i - \sum_{k \in K} \min_{\substack{m \in M_i \\ m \in M_k}} p_{i,m} \quad \forall i \in I \tag{2}$$

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 3-index binary variables (order, unit, event point). The second is tighter, CT4I, due to the explicit consideration of combined processing and changeover tasks and the use of 4-index binaries (order, order, unit, event point).

3-Index Binary Variables Model (CT3I)

A simple example is given to illustrate how CT3I accounts for the sequence-dependent changeover times, see Figure 3. An indirect procedure is employed that starts by calculating the maximum changeover times ($cl_{i,m}^{\max}$) and the difference to the actual changeover ($cl_{i,i',m}^{\Delta}$) through eqs 3-4. Assigning order *i* to unit *m* brings a positive contribution to the slot duration, equal to $p_{i,m}+cl_{i,m}^{\max}$, which considers the worst case scenario in terms of changeovers. To get the true changeover time, one must subtract $cl_{i,i',m}^{\Delta}$ 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). In order to reduce solution degeneracy it is convenient to force orders to be assigned from the last to the first event point.

$$cl_{i,m}^{\max} = \max_{i' \in I_m} cl_{i,i',m} \quad \forall m \in M, i \in I_m \tag{3}$$

$$cl_{i,i',m}^{\Delta} = cl_{i,m}^{\max} - cl_{i,i',m} \quad \forall m \in M, i, i' \in I_m \tag{4}$$

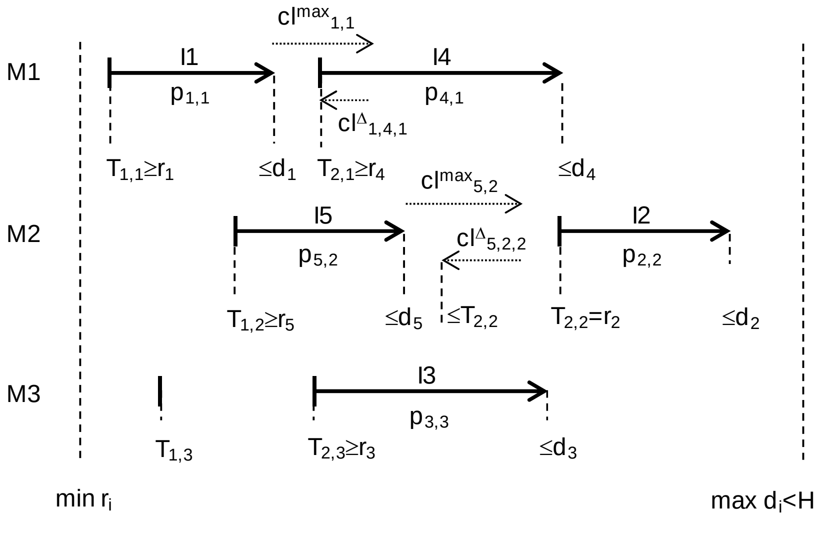


Figure 3. In formulation CT3I, changeovers are calculated through an indirect procedure (|K|=1).

The constructive scheduling formulation using CT3I as the underlying mathematical formulation is illustrated in Figure 4 for the case of one order firstly considered per iteration (*NOS*=1). We use an example consisting of 4 units (two per stage) and rely on the concept of unit-specific time slots meaning that events from different units are affiliated to different time grids. In particular, we can assume[3] that a single slot per order is enough to account for all valid alternatives within the constrained scenario. Since 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.

In the first iteration (*j*=1), there is a single blue order under consideration so it suffices postulating one slot for each of the four time grids. Let us assume that the solver decides to assign the order to M1 and M4 (indicated by the arrows). In the second iteration a red order comes along, and it is no longer possible for the previously assigned blue order to change equipment units. As a consequence, the time grids no longer look the same for *j*=2. While idle units remain with a single time slot, one interval is added to the grids of M1 and M4 to account for the production of red before or after the blue order. One can see the possible slot assignments, with filled rectangles showing previously scheduled orders. In particular, they are placed from right to left in order to reduce solution degeneracy. Now, the solver has decided to assign the red order to M2 and M4, placing it before the blue order in the latter unit.

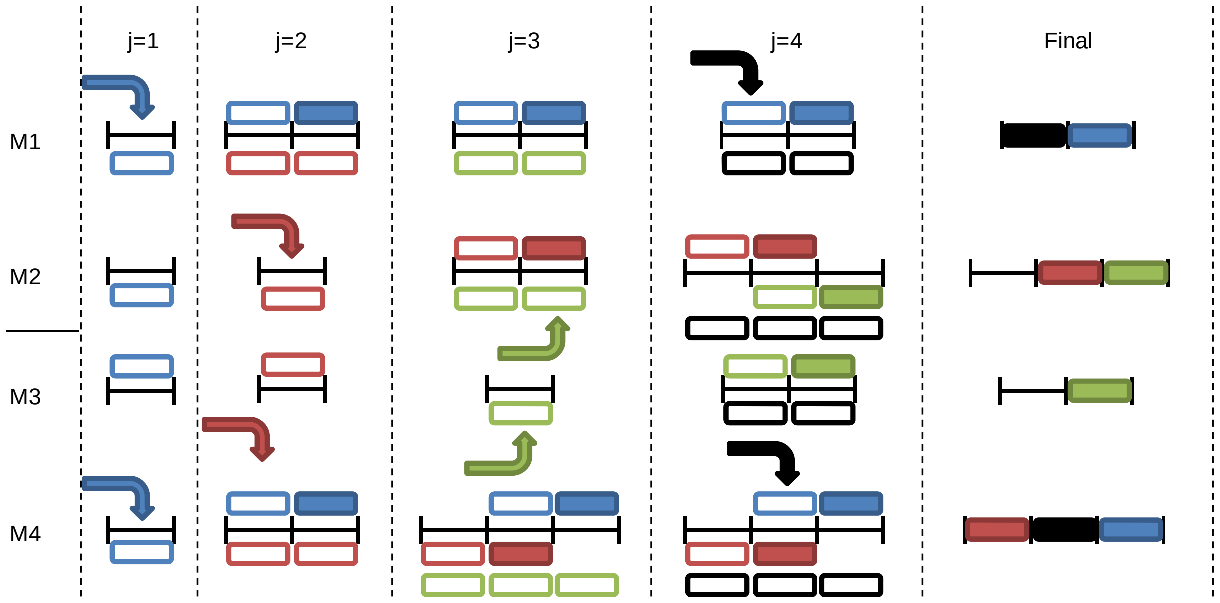


Figure 4. Illustration of constructive scheduling algorithm for model CT3I and one order at a time (NOS=1).

One can see that when going into a particular iteration the number of postulated slots for a given unit is equal to the number of already assigned orders plus the chosen *NOS* value. In the third iteration (*j*=3) there are three slots in M4 to accommodate the blue and red orders and eventually the new green order. Notice that only the latter can be assigned to any slot since we want to keep the relative position of previously assigned orders unchanged. Thus, the blue order cannot start before the second slot so that one free slot is left for the red order (recall that the outcome of *j*=2 is red before blue), neither can red be executed in the final slot. An additional order (black) is included to end the illustration with iteration *j*=4.

The same concept can be applied to other values of parameter *NOS* as can be seen in Figure 5 for the exact same set of orders and *NOS*=2. There are significantly more decisions to be made per iteration, thus leading to more complex problems, but half the number of iterations. Notice that as soon as two orders get assigned to a unit (e.g. *j*=1 for M4) the relative position of orders may change somewhat (blue in slot #2 and red in slot #3 makes it possible having blue before red in *j*=2).

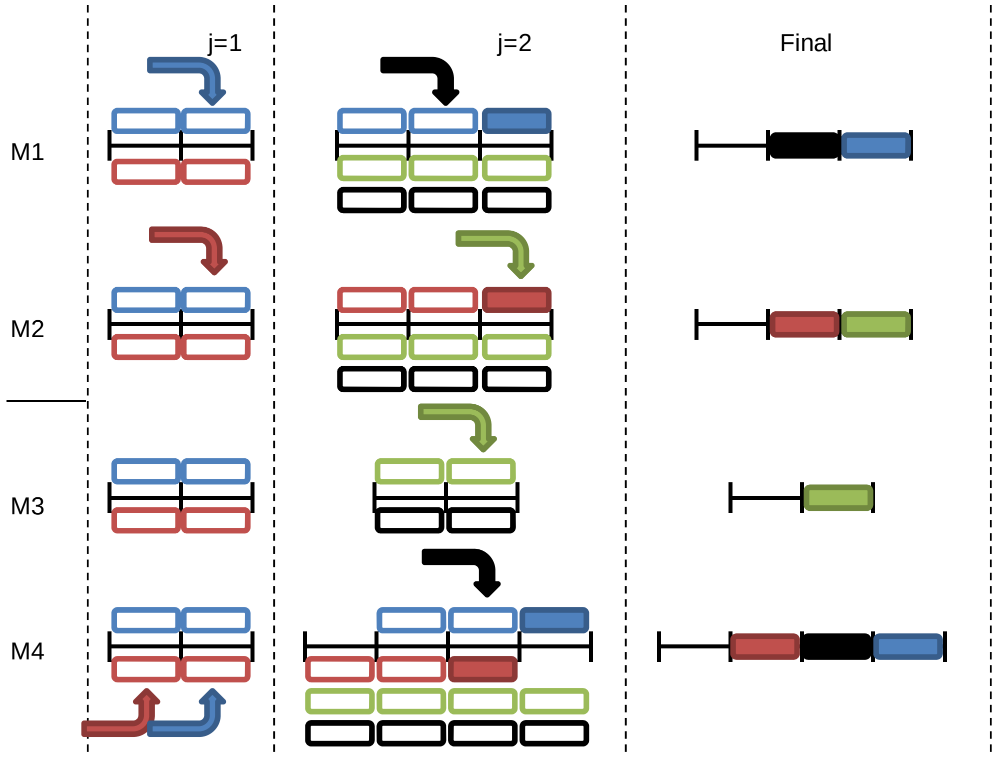


Figure 5. Illustration of constructive scheduling algorithm for model CT3I and NOS=2.

4-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). Note that the last task to be executed typically features a single order index, i.e. (*i*,*i*,*m*), to avoid spending time on changeovers (we assume $cl_{i,i,m}$=0). While in the full-space model certain constraints force, for performance reasons[10], one such task to be the last one executed on a given unit, this cannot be done for the decomposition strategy, so the last task to be executed on non-limiting units might be of type (*i*,*i*',*m*) with *i*≠*i*'. Like with CT3I, postulating a single time slot per order is enough to consider all alternatives[3].

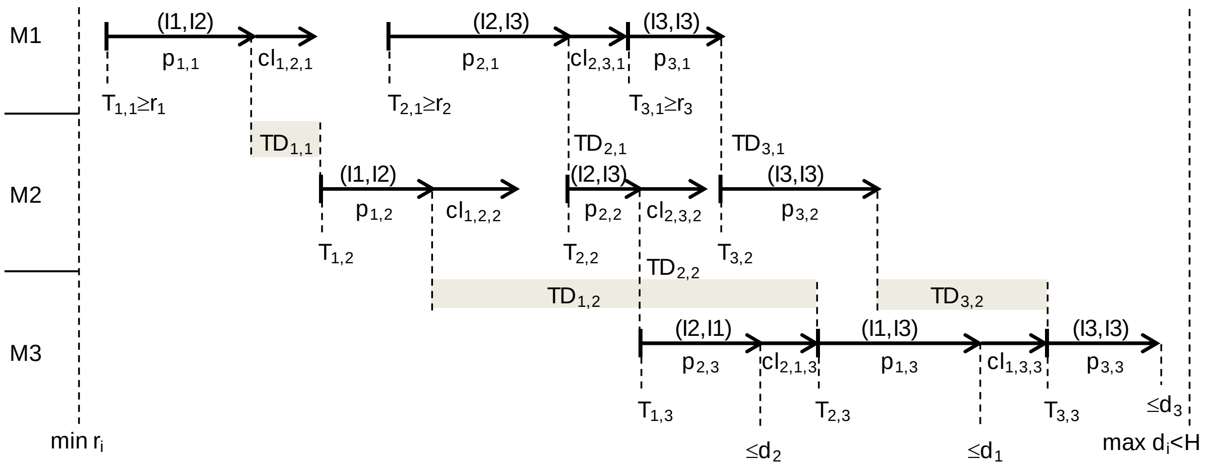


Figure 6. Formulation CT4I models changeovers explicitly through combined processing and changeover tasks (|K|=3).

The first iteration (*j*=1) of the constructive scheduling algorithm is exactly the same as before when tackling one order at a time (*NOS*=1), see Figure 7. There are still no orders assigned to the units and so generality under the constrained scenario is ensured by postulating a single time slot per unit, with the candidate tasks for assignment being of type (*i*,*i*,*m*), e.g. combined tasks (blue, blue). The differences start to appear in the second iteration (*j*=2), where now the previously assigned blue order can only be executed in a single time slot, the second. In general, and depending on the order's position in the sequence from the previous iteration ($pos_{i,m}$), it will be assigned to slot number $pos_{i,m}$×(*NOS*+1). In units M1 and M4, the red order can be either assigned before or after the blue order, which explains the use of three time slots, as proposed in the fixed relative positions strategy of Castro et al.[21] The novelty here is that due to the sequence-dependent changeovers, there is more than one suitable task for previously assigned orders. In the event the red order is executed before the blue order, the appropriate combined task is (blue, blue), otherwise it is (blue, red). For the new red order, there is a single appropriate task on each slot, (red, blue) in slot #1, and (red, red) in slot #3.

Things become increasingly more complicated as we proceed through the iterations. In iteration *j*=3, unit M4 requires five slots in the event the green order gets assigned to it. Again, we need to consider *NOS*+1 alternative tasks for the previously assigned orders, in their fixed time slots. Since red was assigned before blue, it is sufficient to consider (red, blue), (red, green), (blue, green) and (blue, blue). Notice 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); (c) red-blue-green results from (red, blue), (blue, green) in slot #4 and (green, green) in #5. The procedure continues until the final schedule is obtained. Notice that contrary to Figure 4 for *CT3I*, there may be idle slots between orders or empty last slots. Overall, the number of possible assignments (binary variables) for both models is the same per iteration: 4, 10, 16, 22, ..., despite the larger number of time slots being postulated for *CT4I*.

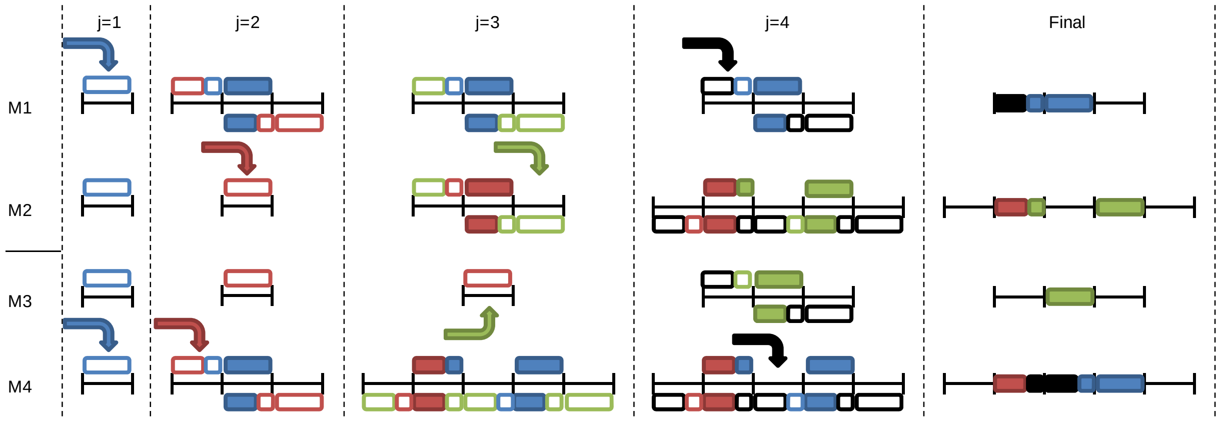


Figure 7. Illustration of constructive scheduling algorithm for model CT4I and NOS=1.

For two orders at a time (*NOS*=2), the number of possible assignments continues to be the same as for *CT3I*, but contrary to *NOS*=1 one may already encounter tasks with different order indices, see Figure 8. The main differences start with iteration *j*=2, where certain combined tasks of to-be-scheduled orders no longer have a single appropriate time slot. Consider task (black, green) as an example. In unit M1 it has to be executed in slots #1 and #4 since 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 machine, a certain sequence can only be obtained with a single combination of tasks. Overall, we get 44 possible assignments for *j*=2 vs. 36 for *CT3I* and the difference will grow with both *NOS* and *j*.

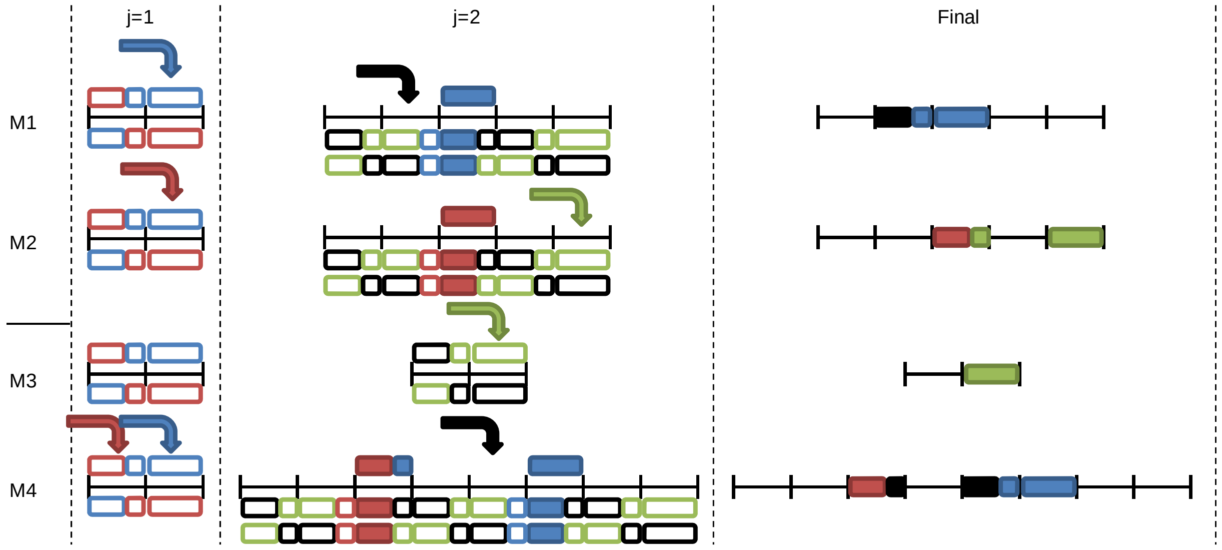


Figure 8. Illustration of constructive scheduling algorithm for model CT4I and NOS=2.

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 papers[9-10,22] 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 3-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

$TD_{i,k}$. The remaining variables are the only ones involved in the objective function Eq. 5. *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. These are penalized in the objective function through weights α and β.

Equation 6 ensures that at most one order is allocated to unit *m* during slot *t*. In order 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), as given in Eq. 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. Equations 9-10 are the release and due date constraints. In cases where the due dates cannot be met, Eq. 10 is relaxed through slack variable $S^1_{t,m}$. Due to the form of the constraint, the slacks are needed for all pairs (slot, unit) and not just for those belonging to last stage units, i.e. the ones penalized in the objective function (Eq. 5). 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 (Eqs. 11-12). The transfer times for orders not involved in stage *k* equal those in the previous stage (Eq. 13). All orders need to be executed once on every defined stage (Eq. 14).

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

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

$$R_{m,t} \le R_{m,t-1} \quad \forall m \in M, t \in T_m^{deg} \tag{7}$$

$$T_{t+1,m}\Big|_{t \notin T_m^{last}} + MS\Big|_{t \in T_m^{last}} - T_{t,m} \ge \sum_{i' \in I_{m,t}} N_{i',m,t} p_{i',m} + \left(N_{i,m,t} cl^{\max}_{i,m} - \sum_{\substack{i' \in I_{m,t+1} \\ i' \ne i}} N_{i',m,t+1} cl^{\Delta}_{i,i',m}\right)\Bigg|_{t \notin T_m^{last}} \tag{8}$$

$$\forall m \in M, t \in T_m^{act}, i \in I_{m,t}$$

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

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

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

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

$$TD_{i,k} = TD_{i,k-1} \ \forall k \in K, i \in I^{act}, i \notin I_k \tag{13}$$

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

Although not required, just like Eq. 7, previous studies[9] have found the following constraints to significantly improve the performance of the full-space model. Equations. 15-16 act as lower and upper bounds on the transfer times. Notice the use of slack variable S_i^2, which allows the due date of order *i* to be violated. Since variables $TD_{i,k}$ have considerable more freedom than their $T_{t,m}$ counterparts, it is more likely for slacks $S_{t,m}^1$ to be active so one should set $\alpha >> \beta$. Finally, Eqs. 17-18 are responsible for reducing the integrality gap.

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

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

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

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

CT4I

Formulation *CT4I* features 4-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 to *CT3I*, the additional index facilitates the writing of the timing constraints linking two consecutive event points and makes other constraints tighter due to 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 since 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], since 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 paper. Eq. 20 then limits the initial states of units to a single order.

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

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

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 to *CT3I*. Equations 22-23 are the release and due date constraints, where the upper bound on the starting time of combined task (*i*,*i'*,*m*), $ub_{i,i'm}$, is calculated through Eq. 24. The two sets of timing variables $T_{t,m}$ and $TD_{i,k}$ are related through Eqs. 25-26. 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. Equation 27 together with Eqs. 5 and 13 (shared with *CT3I*) complete the set of mandatory constraints. 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 with the makespan.

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

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

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

$$ub_{i,i',m} = \min(d_i - \sum_{k\in K_m}\sum_{\substack{k'\in K\\ k'>k}}\min_{\substack{m'\in M_{k'}\\ m'\in M_i}} p_{i,m'} - p_{i,m}, d_{i'} - \sum_{k\in K_m}\sum_{\substack{k'\in K\\ k'>k}}\min_{\substack{m'\in M_{k'}\\ m'\in M_{i'}}} p_{i',m'} - p_{i,m} - cl_{i,i',m} - p_{i',m}\Big|_{i\neq i'}) \tag{24}$$
$$\forall i,i'\in I, m\in M_i\cap M_{i'}$$

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

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

$$\sum_{m\in M_k}\sum_{t\in T_m^{act}}\sum_{\substack{i'\in I_m\\ i\in I_{i',m,t}}} \overline{N}_{i,i',m,t} = 1\ \forall k\in K, i\in I^{act}\bigcap I_k \tag{27}$$

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

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

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

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

Sequencing Variables Model (SV)

One can use the sequencing concept instead of time grids to schedule multistage plants. In particular, models relying on general precedence sequencing variables tend to be more efficient than their immediate precedence counterparts. The drawback is that they are not entirely rigorous since 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 sequencing variables over time grid based models are: (i) they need to be solved only once since 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[21] for rescheduling purposes where large amounts of computational time are seldom available.

We rely on the continuous-time model of Harjunkoski and Grossmann[22] 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. 32-33). Notice that the sequencing variables are only relevant if both i and i' are assigned to the same unit. 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 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.[21]

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

$$\begin{aligned}&Tf_{i,k} \geq Tf_{i',k} + p_{i,m} + cl_{i',i,m} \times (1 - X_{i,i',k}) - \max_{i''\in I^{active}} d_{i''} \times (2 + X_{i,i',k} - Y_{i,m} - Y_{i',m})\\ &\forall i,i' \in I^{act}, i'>i, k\in K, m\in M_k \cap M_i \cap M_{i'}\end{aligned} \quad (33)$$

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

Scheduling Algorithm

The proposed algorithm comprises two parts[21]. 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 so called 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* that gives the number of orders per iteration that are scheduled with the full degree of freedom; (ii) distribution of orders over iterations, following a pre-ordering heuristic, or a random choice and given by set I_j; (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 coming 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 entire 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 $pos_{i,m}$ can be easily calculated based on the value of the binary variables (see Castro et al.[21] 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.[21].

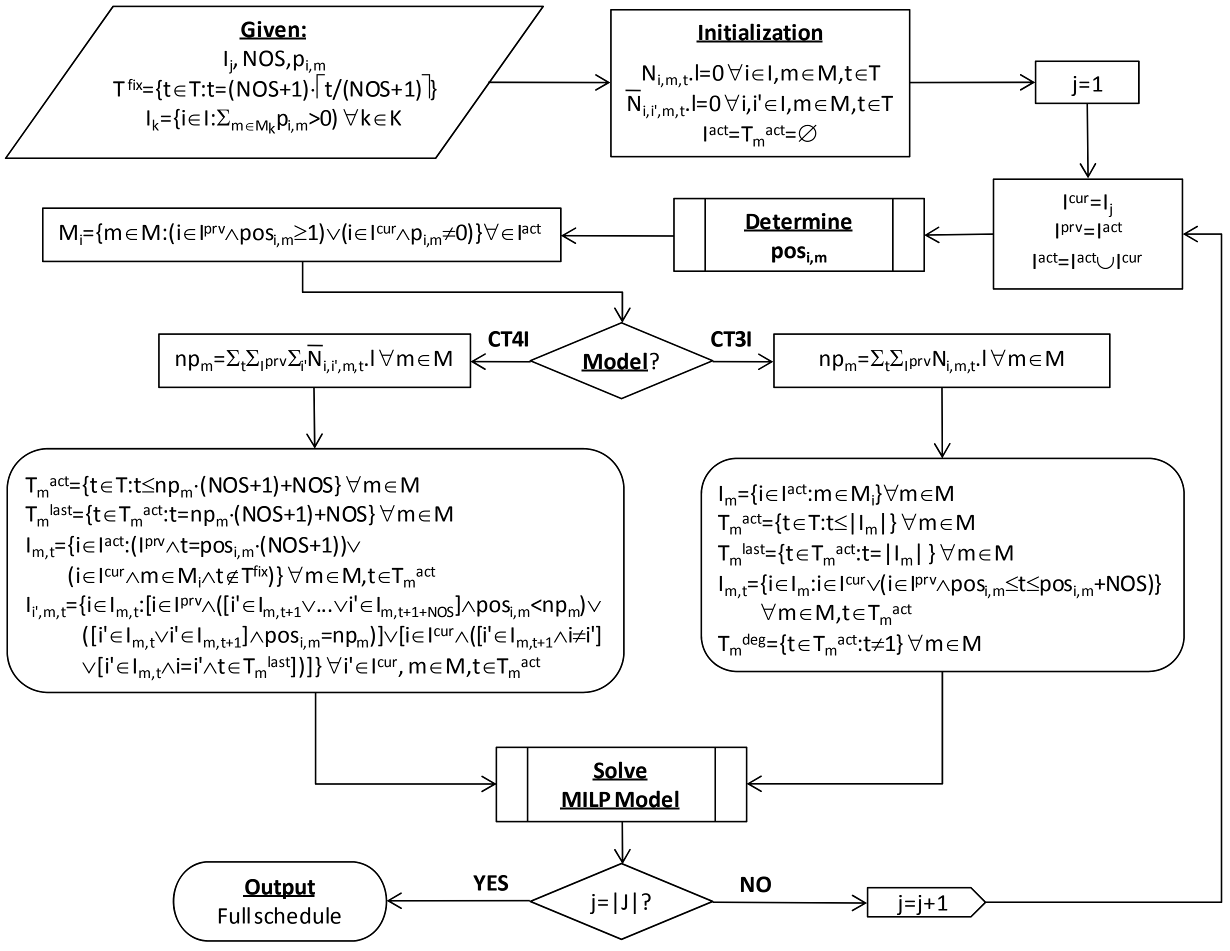

Given:
I_j, NOS, $p_{i,m}$
$T^{fix}=\{t\in T: t=(NOS+1)\cdot\lceil t/(NOS+1)\rceil\}$
$I_k=\{i\in I:\Sigma_{m\in M_k} p_{i,m}>0)\ \forall k\in K$
Initialization
$N_{i,m,t}.l=0\ \forall i\in I, m\in M, t\in T$
$\overline{N}_{i,i',m,t}.l=0\ \forall i,i'\in I, m\in M, t\in T$
$I^{act}=T_m^{act}=\varnothing$
j=1
$I^{cur}=I_j$
$I^{prv}=I^{act}$
$I^{act}=I^{act}\cup I^{cur}$
Determine $pos_{i,m}$
$M_i=\{m\in M:(i\in I^{prv}\wedge pos_{i,m}\geq 1)\vee(i\in I^{cur}\wedge p_{i,m}\neq 0)\}\forall\in I^{act}$
Model?
CT4I
CT3I
$np_m=\Sigma_t\Sigma_{I^{prv}}\Sigma_{i'}\overline{N}_{i,i',m,t}.l\ \forall m\in M$
$np_m=\Sigma_t\Sigma_{I^{prv}}N_{i,m,t}.l\ \forall m\in M$
$T_m^{act}=\{t\in T: t\leq np_m\cdot(NOS+1)+NOS\}\ \forall m\in M$
$T_m^{last}=\{t\in T_m^{act}: t=np_m\cdot(NOS+1)+NOS\}\ \forall m\in M$
$I_{m,t}=\{i\in I^{act}:(I^{prv}\wedge t=pos_{i,m}\cdot(NOS+1))\vee$
$(i\in I^{cur}\wedge m\in M_i\wedge t\notin T^{fix})\}\ \forall m\in M, t\in T_m^{act}$
$I_{i',m,t}=\{i\in I_{m,t}:[i\in I^{prv}\wedge([i'\in I_{m,t+1}\vee\ldots\vee i'\in I_{m,t+1+NOS}]\wedge pos_{i,m}<np_m)\vee$
$([i'\in I_{m,t}\vee i'\in I_{m,t+1}]\wedge pos_{i,m}=np_m)]\vee[i\in I^{cur}\wedge([i'\in I_{m,t+1}\wedge i\neq i']$
$\vee[i'\in I_{m,t}\wedge i=i'\wedge t\in T_m^{last}])]\}\ \forall i'\in I^{cur}, m\in M, t\in T_m^{act}$
$I_m=\{i\in I^{act}: m\in M_i\}\forall m\in M$
$T_m^{act}=\{t\in T: t\leq|I_m|\}\ \forall m\in M$
$T_m^{last}=\{t\in T_m^{act}: t=|I_m|\}\ \forall m\in M$
$I_{m,t}=\{i\in I_m: i\in I^{cur}\vee(i\in I^{prv}\wedge pos_{i,m}\leq t\leq pos_{i,m}+NOS)\}$
$\forall m\in M, t\in T_m^{act}$
$T_m^{deg}=\{t\in T_m^{act}: t\neq 1\}\ \forall m\in M$
Solve MILP Model
j=|J|?
YES
NO
Output
Full schedule
j=j+1

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

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 Figure 7 and Figure 8, any previous order can be assigned to a single fixed slot $t \in T^{fix}$, 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 $np_m \times NOS + NOS$ 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 Figure 7 and Figure 8, which were described earlier.

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 NOS+1 slots, from $pos_{i,m}$ forward, as can be seen in Figure 4 and Figure 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 of the rescheduling algorithm.

Rescheduling

While 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[21] 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.[21] for details concerning the rescheduling algorithm, which is based on the original ideas of Roslöf et al.[14] and Méndez and Cerdá[15]. It relies on the general precedence sequencing variables model SV 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 of the constructive part. Recall that increasing the *NOS* value can potentially lead to better solutions. Keeping I_j 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 paper.

Computational Results

The performance of the scheduling algorithm is now illustrated through the solution of ten 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 paper, as well as by a constraint programming 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 6 stages. It does neither involve release nor due dates. For the former problem set, we have set α=10 and β=1 (Eq. 5) to prioritize schedules that meet the due date constraints. Problems P14-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) 60,000 CPUs for the full-space models; (ii) 3,600 CPUs per iteration on the constructive part; (iii) 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 magnitude savings in computational time. For instance, *SV* ran out of memory at 23408 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. Particularly, one may always expect a feasible outcome, even though solution quality will degrade with the increase in problem size. This is apparent from the results of P14 and P15. While for the former, *SV* was able to find a solution with a makespan equal to 36.121 before running out of memory at 51900 CPUs, a value 16% higher than the best solution found by the decomposition algorithm in considerably less time (see Table 2), for P15, the returned makespan after 16 hours of computational time (55.220) is 30% higher than the best.

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

Problem	Features (|I|,|M|,|K|)	Optimum	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	24105[*]

[*]Out of memory termination, suboptimal solution found=275, best possible=259.

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. Since 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) due to 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 paper to further explore this aspect.

While the influence of the model in the 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 to outcomes with no due date violations obtained in problems of similar size but without changeovers[21]. 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.

Table 2. Solution from Scheduling Algorithm for Different Model Choices and NOS Values (Best solution in Bold, Solution with Violation of Due Dates in Italic)

	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**

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 near optimal solutions, the other is obtaining them in small computational times. From the total computational efforts listed in Table 3, one can see that reasonably good solutions can be obtained up to roughly 6 minutes of computational time when using *CT4I* or *SV* with *NOS*=1. 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, highlighting once more the importance of having a set of competitive models rather than a single one. 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-P15 in less than 1 hour. The solutions then degrade for *NOS*=3, primarily because we can no longer solve all subproblems to optimality up to the 1-hour maximum resource limit. This effect is not noticeable for *SV*, probably due to 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 nearly 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, since the increased difficulty per problem is somewhat compensated by the need to solve fewer of them (see Eq. 1).

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

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[23] found solutions with a makespan equal to 30.480 and 50.721 h for P14 and P16, respectively. More recently, Kopanos and Puigjaner[24] have shown a 29.91 h schedule in the poster session of ESCAPE-19. We did slightly better with *CT4I*, *NOS*=2, and a different ordering heuristic for rescheduling, before upgrading from CPLEX 11.1, 29.871 h. 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 Supplementary Material, the reader can possibly find better solutions.

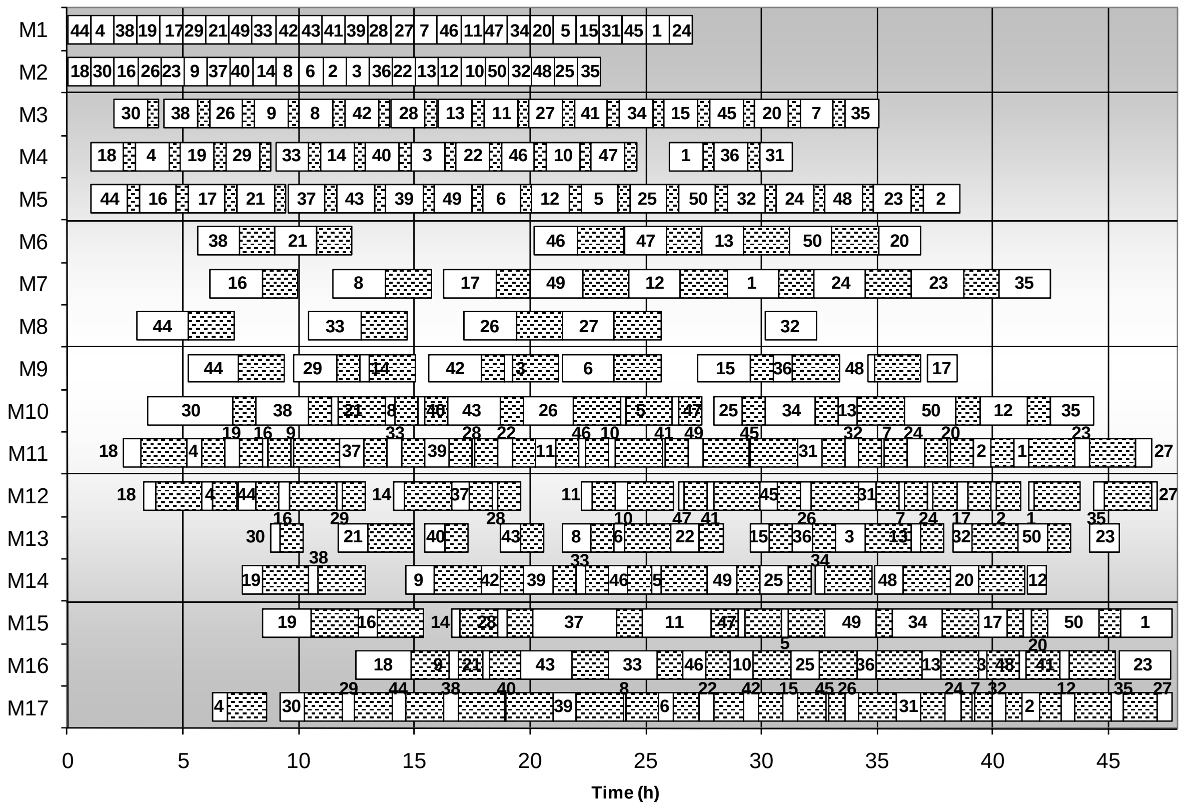


Figure 10. Best found solution for P16 (makespan=47.722 h).

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 programming formulations, with three conceptually different options being available to the user. While a 3-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 4-index unit-specific formulation and with a sequencing variables 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.

References

(1) Grossmann IE. Enterprise-wide Optimization: A New Frontier in Process Systems Engineering. *AIChE J*. **2005**, *51*, 1846.

(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.

(3) Harjunkoski I, Nyström R, Horch A. Integration of Scheduling and Control- Theory of Practice? *Comput. Chem. Eng.* **2009**, doi:10.1016/j.compchemeng.2009.06.016.

(4) Floudas CA, Lin X. Continuous-time versus Discrete-time Approaches for Scheduling of Chemical Processes: A Review. *Comput. Chem. Eng*. **2004**, *28*, 2109.

(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.

(6) Wassick J. Enterprise-wide Optimization in an Integrated Chemical Complex. *Comput. Chem. Eng*. **2009**, 33, 1950.

(7) Henning G. Production Scheduling in the Process Industries: Current Trends, Emerging Challenges and Opportunities. In Computer Aided Chemical Engineering, Vol 27. Editors: Rita Alves, Claudio Nascimento and Evaristo Biscaia Jr., Elsevier, 23.

(8) Pantelides CC. Unified Frameworks for the Optimal Process Planning and Scheduling. In 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.

(10) Castro PM, Novais AQ. Scheduling Multistage Batch Plants with Sequence-Dependent Changeovers. AIChE J. **2009**, 55, 2122.

(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) Roslöf J, Harjunkoski I, Björkqvist J, Karlsson S, Westerlund T. An MILP-based reordering algorithm for complex industrial scheduling and rescheduling. *Comp. Chem. Eng*. **2001**, *25*, 821.

(15) Méndez C, Cerdá J. Dynamic scheduling in multiproduct batch plants. *Comp. Chem. Eng*. **2003**, *27*, 1247.

(16) Dimitriadis AD, Shah N, Pantelides CC. RTN-Based Rolling Horizon Algorithms for Medium Term Scheduling of Multipurpose Plants. *Comput. Chem. Eng.* **1997**, *21*, S1061.

(17) 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.

(18) 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.

(19) 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.

(20) Ierapetritou MG, Floudas CA. Effective Continuous-Time Formulation for Short-Term Scheduling. 1. Multipurpose Batch Processes. *Ind. Eng. Chem. Res.* **1998**, *37*, 4341.

(21) Castro PM, Harjunkoski I, Grossmann IE. Optimal Short-Term Scheduling of Large-Scale Multistage Batch Plants. Ind. Eng. Chem. Res. In press.

(22) Harjunkoski I, Grossmann IE. Decomposition Techniques for Multistage Scheduling Problems using Mixed-integer and Constraint Programming Methods. Comput. Chem. Eng. **2002**; 26: 1533-1552.

(23) Castro P, Méndez C, Grossmann I, Harjunkoski I, Fahl M. Efficient MILP-based Solution Strategies for Large-Scale Industrial Batch Scheduling Problems. In Computer Aided Chemical Engineering, Vol 21. Editors: Wolfgang Marquardt and Costas Pantelides, Elsevier, 2231.

(24) Kopanos G, Puigjaner L. A MILP Scheduling Model for Multi-Stage Batch Plants. In Computer Aided Chemical Engineering, Vol 26. Editors: Jacek Jezowski and Jan Thullie, Elsevier, 369.