A Hybrid MILP/CP Decomposition Approach for the Continuous Time Scheduling of Multipurpose Batch Plants

Christos T. Maravelias, Ignacio E. Grossmann[*]
Carnegie Mellon University, Department of Chemical Engineering
Pittsburgh, PA 15213

Submitted March 2003
Revised Manuscript Submitted December 2003

Abstract

A hybrid Mixed-Integer Linear Programming (MILP)/Constraint Programming (CP) decomposition algorithm is proposed for the short-term scheduling of batch plants that rely on the State Task Network representation. The decisions about the type and number of tasks performed, as well as the assignment of units to tasks are made by the MILP master problem. The CP subproblem checks the feasibility of a specific assignment and generates integer cuts for the master problem. A graph-theoretic preprocessing that determines time windows for the tasks and equipment units is also performed to enhance the performance of the algorithm. To exclude as many infeasible configurations as possible, three classes of integer cuts are generated. Various objective functions such as the minimization of assignment cost, the minimization of makespan for fixed demand and the maximization of profit for a fixed time horizon can be accommodated. Variable batch-sizes and durations, different storage policies, and resource constraints are taken into account. The proposed framework is very general and can be used for the solution of almost all batch scheduling problems. Numerical results show that for some classes of problems, the proposed algorithm can be two to three orders of magnitude faster than standalone MILP and CP models.

1. Introduction

The problem of short term scheduling of general (or multipurpose) batch plants has received considerable attention in the literature. Kondili et al. (1993) proposed the discrete-time State Task Network (STN) representation, and Shah et al. (1993) proposed a reformulation. The equivalent Resource Task Network (RTN) representation was proposed by Pantelides (1994). Continuous time representations, which are more general but potentially difficult to solve, were proposed by Sargent and Zhang (1995), and Mockus and Reklaitis (1999). Based on the STN and RTN representations, several other continuous time MILP models have been proposed since then (e.g. Ierapetritou and Floudas, 1998; Lee et al., 2001; Castro et al., 2001; Maravelias and Grossmann, 2003a). Although the performance of these models in several test problems has proved to be encouraging, their major limitation is that for large problems they can become computationally expensive due to the big-M constraints that are used for time matching. This has led previous workers to consider special cases of the general multipurpose batch plant problem. Some of the common assumptions are, (a) no resource constraints other than equipment, and (b) no batch splitting and mixing. Furthermore, the computational performance of STN- and RTN-based models is poor when the objective is the minimization of makespan for a given demand. For short-term scheduling, however, the demands are usually fixed, which implies that the minimization of makespan or the minimization of production cost is often a more realistic objective.

Constraint Programming (CP) is a new modeling and solution paradigm that has proved to be very effective for solving certain classes of problems (van Hentenryck, 1989; Marriot and Stuckey, 1999; Hooker, 2000).

[*] To whom all correspondence should be addressed:
E-mail: grossmann@cmu.edu, Tel: 412-268-3642, Fax: 412-268-7139

Constraint Programming is particularly effective for solving feasibility problems and seems to be better than traditional MILP approaches in discrete optimization problems where finding a feasible solution is difficult. The lack of an obvious relaxation, however, makes CP worse for loosely constrained problems, where the focus is on finding the optimal solution among many feasible ones and proving optimality. Constraint Programming has been successfully used for some classes of scheduling problems (Baptiste et al., 2001).

In the general short-term scheduling of multipurpose batch plants the number of tasks (jobs) is not known a priori: it is a decision variable that is to be determined by the optimization. Moreover, in its general form (variable processing times and batch sizes, recycle streams, batch splitting/mixing, resources other than equipment, changeover times) the scheduling problem is loosely constrained and a standalone CP approach is not efficient. Thus, in this work we have combined the traditional MILP approach with the CP approach in order to take advantage of their complementary strengths. We use MILP to optimize and find partial solutions and CP to check feasibility and produce complete solutions. Similar approaches have been proposed by Jain and Grossmann (2001), and Harjunkoski and Grossmann (2002) for the restricted cases of minimization of assignment cost of single and multistage batch plants, respectively, yielding considerable computational improvements compared to standalone MILP and CP models.

The proposed hybrid MILP/CP method consists of two phases and is based on the MILP model by Maravelias and Grossmann (2003a). In the first phase we determine the earliest start times (EST) and the latest finish times (LFT) of tasks and units, and derive strong integer cuts that exclude infeasible "subconfigurations." In the second phase we use an iterative scheme where we solve a master MILP model and a CP subproblem. The master problem yields a set of tasks to be performed and their assignments to units. The subproblem checks feasibility and derives a feasible schedule, if one exists, for the assignment obtained by the master problem. Integer cuts are added in the master problem to exclude the current assignment. Three objective functions are considered: (a) minimization of makespan subject to satisfying a given demand, (b) maximization of profit for a fixed time horizon, and (c) minimization of cost subject to satisfying given orders with due dates. Numerical examples are presented that show that orders of magnitude reduction in computational effort can be achieved with the proposed hybrid approach.

2. Background

2.1. Mixed-Integer Formulations for Multipurpose Batch Plants

In this paper we address the scheduling of multipurpose batch plants. An example is shown in Figure 1, where raw material S1 is heated to form intermediate S2, which is then used for the production of intermediate S3 (Reaction 1), intermediate S4 and final product S5 (Reaction 2). Intermediates S3 and S4 are mixed to produce final product S7 (Reaction 3) and intermediate S3 is purified to give final product S6 and S2 that is recycled (Separation). Multipurpose batch plants exhibit a number of features that make their modeling challenging. As shown in Figure 1, unlike traditional scheduling problems where each batch can be viewed as an entity moving throughout the plant, we may have batch splitting (e.g. a batch of S2 can be used for Reaction 1 and Reaction 2), batch mixing (e.g. intermediates S3 and S4 are mixed to produce one batch of S7) or recycle streams. Thus, general batch plants can be viewed as the generalization of all possible plant configurations. Furthermore, the duration and the batch size of the tasks may not be constant. Moreover, resource constraints other than those on equipment units may be present (e.g. manpower, cooling water, etc.), and different storage policies (UIS/FIS/NIS/ZW) can be applied for the various chemicals. In order to account for all these issues, we need to monitor the level of inventories and the level of resource consumption throughout the time horizon. To do so, we need to partition the time horizon into a sufficiently large number of periods and enforce equipment unit, utility and inventory feasibility for all time periods, which leads to large and difficult to solve formulations.

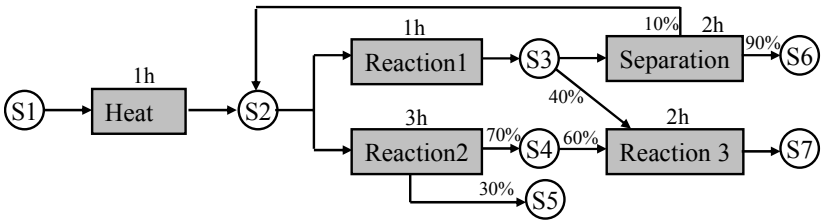


Figure 1: Multipurpose batch plant

Kondili et al. (1993) formulated the STN representation for batch scheduling with an MILP using a discrete-time representation (Figure 2a), where the time horizon is divided into H intervals of equal duration, common for all units, and where the tasks begin and finish at a time point. This means that the duration of the intervals must be equal to the greatest common factor of the processing times of tasks and that the discrete-time representation can only be used when the processing times are constant. The assumption of constant processing times is not always realistic, while the length of the intervals may be so small that it either leads to a prohibitive number of intervals rendering the resulting model unsolvable, or else it requires approximations which may compromise the feasibility and optimality of the solution. It should be noted that the Resource Task Network (RTN) formulation by Pantelides (1994) provides a more compact representation but shares similar limitations as the STN model.

To circumvent the above cited difficulties, two different continuous-time representations have been proposed in which the time horizon is divided into time intervals of unequal and unknown duration, common for all units. In the continuous time representation I (Figure 2b), each task must start and finish exactly at a time point (Zhang and Sargent, 1995; Schilling and Pantelides, 1996; Mockus and Reklaitis, 1999), while in the representation II (Figure 2c), each task must start at a time point but it need not finish at a time point (Castro et al., 2001). In both representations, the number of time points is determined with an iterative procedure, during which the number of time points is increased by one until there is no improvement in the objective function. Since time points are not fixed, constraints that match a time point with the start (or finish) of a task are necessary. These constraints are big-M constraints that result in poor LP relaxations. On the other hand, the continuous-time representation accounts for variable processing times, and is more realistic than the discrete-time representation. It also requires significantly fewer time intervals and hence leads to smaller problems. A combination of the two types of continuous-time representations was proposed by Maravelias and Grossmann (2003a).

An alternative approach is the event-point representation (Figure 2d) proposed by Ierapetritou and Floudas (1998), where time events are not common for all units. In this approach, the time horizon is divided into a number of events which are different for each unit, subject to certain sequencing constraints. Since events need not be common among units, the number of events necessary in this approach is usually smaller than the number of periods required in other continuous-time representations, and the proposed model is faster compared to continuous-time STN formulations with common time grid. However, as has been shown in Maravelias and Grossmann (2003a), this representation may not always yield accurate representations of batch operations.

Finally, a number of papers address special cases of multipurpose batch plants (Rodrigues et al., 2000; Mendez et al., 2000; Mendez and Cerda, 2000; Mendez et al., 2001; Lee et al., 2001). In most of these papers, special assumptions are made to allow the development of special MILP models that are easier to solve. Some of the common assumptions are, (a) the plant has some special configuration, (b) no batch splitting and mixing is allowed, and (c) there are no resource constraints other than those on equipment units.

The objective function in most of these approaches is the maximization of profit for a fixed time horizon. In short-term scheduling, however, the demand is usually fixed and more meaningful objectives are to minimize the makespan for a fixed demand, or minimize the production cost for fixed demands with due dates. As has

been shown, however, the computational performance of STN-based MILP models is poor when the objective is the minimization of makespan. For this reason we consider Constraint Programming as an alternative solution technique.

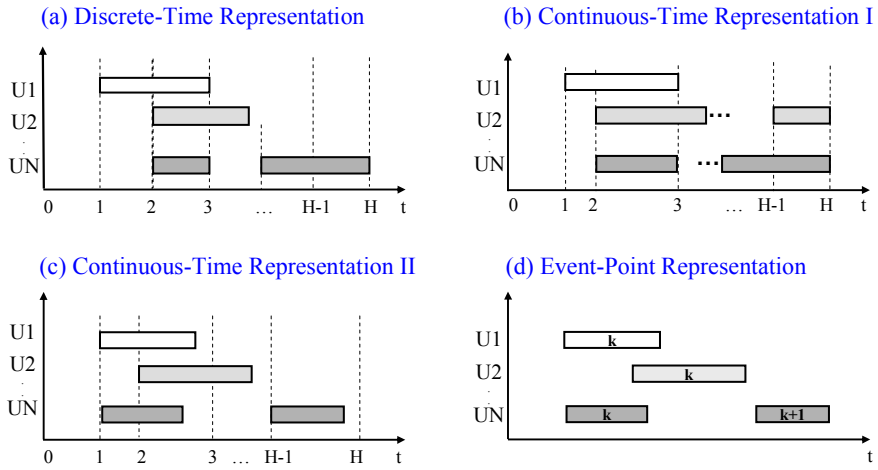


Figure 2: Time Representations.

2.2. Constraint Programming and MILP/CP Integration Schemes

Constraint Programming (van Hentenryck, 1989; Hooker, 2000) was originally developed to solve feasibility problems, but it has been extended to solve optimization problems as well. Constraint programming (CP) is based on performing a tree enumeration by reducing at each node the domains of the variables, which can be continuous, general integer, boolean and binary. If an empty domain is found the domain is pruned. Branching is performed whenever a domain of an integer, binary or boolean variable has more than one element, or when the bounds of the domain of a continuous variable do not lie within a tolerance. Whenever a solution is found, or a domain of a variable is reduced, new constraints are added. The search terminates when no further nodes must be examined. Constraint programming algorithms are very efficient for some classes of problems, among which scheduling is a prominent one. Due to the general nature of the batch plants that appear in chemical industry, however, a standalone CP model would not be effective for solving this class of problems. This is due to the fact that the type and number of tasks to be performed in a general batch plant are optimization decisions, which implies that the number of potential activities in the CP formulation may be very large and, moreover, the large number of alternative production paths makes the CP problem loosely constrained. The computational performance of a standalone CP model that we developed was indeed very poor. Specifically, the computational time required for a well studied example of Kondili et al. (1993), that involves five tasks, four units and nine states, by a standalone CP model is more than 300 CPU sec, whereas the same problem is solved in less than 1 CPU sec by any continuous-time MILP model.

Models that integrate MILP and CP have also appeared recently. The motivation for this integration follows from the fact that MILP and CP have complementary strengths that can be exploited simultaneously. Mathematical programming techniques are efficient in finding optimal solutions and providing good bounds, while the CP language is more expressive and the CP search techniques are often more efficient in solving feasibility problems. The integration between MILP and CP can be achieved in two ways (Hooker, 2002; van Hentenryck, 2002):

(a) By combining MILP and CP constraints into one hybrid model. In this case a hybrid algorithm is also needed for the solution of the model.
(b) By decomposing the original problem into two subproblems: one MILP and one CP problem. Each model is solved separately and information obtained while solving one subproblem is used for the solution of the other subproblem.

More information on CP can be found in Marriot and Stuckey (1999) and Hooker (2000). Constrained-based scheduling algorithms can be found in Baptiste et al. (2001), and a short description of MILP/CP integration schemes can be found in Jain and Grossmann (2001).

3. Problem Statement

For the problem addressed in this paper we assume that we are given:
(i) a fixed or variable time horizon
(ii) the available units and storage tanks, and their capacities
(iii) the available utilities and their upper limits
(iv) the production recipe (mass balance coefficients, utility requirements)
(v) the processing times and changeover times
(vi) the amounts of available raw materials; the demand of final products and their due dates
(vii) the prices of raw materials and final products
The goal is then to determine:
(i) the type and number of tasks performed
(ii) the assignment of equipment units to tasks
(iii) the sequencing and timing of tasks taking place in each unit
(iv) the batch size and duration of tasks
(v) the amount of resources allocated to each task
(vi) the amount of raw materials purchased and the amount of final products sold
The objective can be, (a) the maximization of production, income or profit for a fixed time horizon, or (b) the minimization of makespan for a specified demand, or (c) the minimization of cost for specified orders with due dates.

4. Proposed Hybrid Algorithm

As mentioned above, several STN-based MILP models have been proposed for the scheduling of batch plants. Discrete-time models, while computationally more effective than the continuous-time models, often require approximations that may give infeasible or suboptimal solutions. Furthermore, the scheduling of medium complexity process networks (10-20 tasks, 10-20 states) becomes intractable when the number of intervals is above 60. Continuous-time and event-based models, while more general in terms of task durations, become computationally intractable even more quickly. Specifically, medium-complexity STN networks, problems with more than 15 intervals are intractable. Another shortcoming of continuous-time STN models is that they perform reasonably well only for the maximization of profit over a fixed time horizon. Their computational performance for other objectives is often poor (Maravelias and Grossmann, 2003a).

The difficulty in solving STN scheduling problems led us to develop a hybrid MILP/CP method that exploits the complementary strengths of MILP and CP. We use MILP to optimize the high level decisions, and CP to determine a feasible detailed schedule. Specifically, we propose an iterative scheme where we iterate between a MILP master problem and a CP subproblem, in a similar fashion as in Jain and Grossmann (2001). However, in this work the type and number of tasks to be performed and the assignment of tasks to equipment units are determined in the master MILP problem, while the CP subproblem is used to derive a feasible schedule for the assignment obtained by the master problem. At each iteration, one or more specialized integer cuts are added to the master problem to exclude infeasible or previously obtained assignments. For a maximization problem, the relaxed master problem provides an upper bound and the

subproblem, when feasible, provides a lower bound. To enhance the performance of the algorithm, preprocessing is performed before the main iterative scheme. Preprocessing is used to determine Earliest Start Times (EST) and Latest Finish Times (LFT) of both tasks and units, and to create strong integer cuts that are added a priori in the cut-pool of the master problem. Moreover, the proposed method can be used to obtain more than one feasible solutions; we just need to, (a) store all feasible solutions found during the execution, and (b) keep iterating after the convergence of the bounds. For the implementation of the method we used ILOG's OPL Studio 3.5. (Appendix A). A simplified flow diagram of the proposed algorithm for the maximization of profit is shown in Figure 3. The flow diagrams for the minimization of makespan and cost are similar.

4.1. MILP Master Problem

For the master MILP problem an aggregated STN representation has been used with no time points, or equivalent continuous time intervals. Only assignment, batch size and mass balance constraints are included, and since there are no time points, mass balance constraints are expressed once (for the total amounts) at the end of the time horizon for each state. Special integer cuts are added to exclude previously found sets of tasks. Resource constraints other than equipment are not considered.

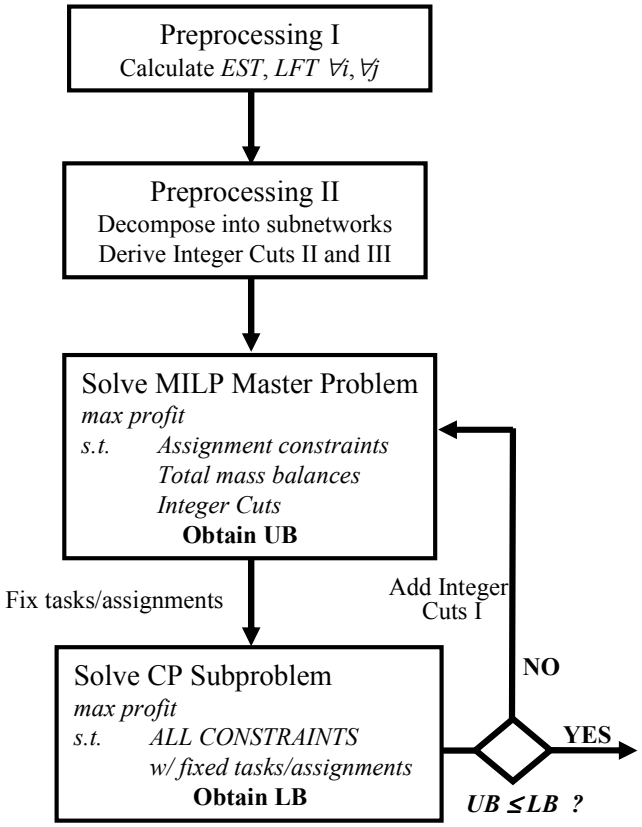


Figure 3: Schematic diagram of the proposed hybrid MILP/CP algorithm.

In order to decouple units from tasks, we use the following rule: if a task *i* can be performed in both units *j* and *j'*, then two tasks *i* (performed in unit *j*) and *i'* (performed in unit *j'*) are defined (see Ierapetritou and Floudas, 1998). As explained above, the number of times each task is carried out is to be determined by the optimization, and thus for each task *i* we postulate a maximum number of copies, i.e. an upper bound on the number of batches of task *i* that can be carried out in any feasible solution. A strict upper bound C_i^{MAX} on the

number of copies of task i is given by equation (1) where H is an upper bound on the length of the time horizon and D^{MIN} is the minimum duration of task i. In practice, though, we can use our knowledge about the process network to postulate a smaller number of copies.

$$\left|C_i\right| = C_i^{MAX} = \left\lfloor H / D^{MIN} \right\rfloor \quad \forall i \tag{1}$$

For each copy c of task i we define the binary Z_{ic}, which is equal to 1 if the c^{th} copy of task i is carried out. We also define its duration D_{ic} and batch size B_{ic}. For each state, we define its inventory level $\hat{S}_s$ at the end of the scheduling horizon. The master MILP problem is shown in Appendix C to be a relaxation of the MILP model of Maravelias and Grossmann (2003a) given in Appendix B. Hence, the master MILP problem (MP) provides an upper bound to the profit and a lower bound to the cost or makespan. It consists of equations (2) to (10):

$$\sum_{i \in I(j)} \sum_{c} D_{ic} \le MS \quad \forall j \tag{2}$$

$$D_{ic} = \alpha_i Z_{ic} + \beta_i B_{ic} \quad \forall i, \forall c \in C_i \tag{3}$$

$$B_i^{MIN} Z_{ic} \le B_{ic} \le B_i^{MAX} Z_{ic} \quad \forall i, \forall c \in C_i \tag{4}$$

$$\hat{S}_s = S0_s + \sum_{i \in O(s)} \sum_{c} \rho_{is}^{O} B_{ic} - \sum_{i \in I(s)} \sum_{c} \rho_{is}^{I} B_{ic} \quad \forall s \tag{5}$$

$$\hat{S}_s \ge d_s \quad \forall s \in FP \tag{6}$$

$$\hat{S}_s \le C_s \quad \forall s \in INT \tag{7}$$

$$Z_{ic+1} \le Z_{ic} \quad \forall i, \forall c \in C_i, c < \left|C_i\right| \tag{8}$$

Integer Cuts (9)

Objective Function (10)

$$Z_{ic} \in \{0,1\},\ D_{ic} \ge 0, B_{ic} \ge 0, \hat{S}_s \ge 0$$

Constraint (2) is a relaxed assignment constraint which enforces that the sum of the durations of the tasks assigned to a unit does not exceed the scheduling horizon *MS*, where *I(j)* is the set of tasks that can be assigned to unit j. The duration of copy c of task i is a function of its batch size [constraint (3)], and the batch size of copy c of task i is bounded through constraint (4). The amount of state s at the end of the time horizon $\hat{S}_s$ is calculated by (5) to be equal to the initial amount $S0_s$ plus the amount produced, minus the amount consumed, where ρ_{is}^{I} and ρ_{is}^{O} are the mass fractions for consumption and production, respectively, of state s by task i. Note that constraint (5) ensures that the net production of state s in non-negative, but since we do not monitor and restrict the inventory level of state s during the entire scheduling horizon, a solution of the master problem may imply that the level of state s is at some point negative. If state s corresponds to a final product ($s \in FP$), $\hat{S}_s$ must be greater than the demand d_s [constraint (6)]; if it corresponds to an intermediate ($s \in INT$) it must be less than the capacity C_s of the storage tank of state s [constraint (7)]. Constraint (8) is used to eliminate symmetric assignments by enforcing the condition that copy $c+1$ of task i can be carried out only if copy c is carried out. At a specific iteration k, constraints (9) include all the integer cuts that have been added during preprocessing and in previous iterations. The objective function, which as noted above provides a bound, can be the maximization of profit for a fixed time horizon, the minimization of makespan for fixed demand, or the minimization of production cost for fixed demand and due dates. The exact form of the objective function is given in section 4.5.

4.2. CP Subproblem

In this paper we model CP subproblems using the modeling language of ILOG's OPL Studio 3.5, which has a number of global constraints and special constructs specifically developed for scheduling applications, for which a short description can be found in Appendix A. The description of the CP subproblem, hence, is made in terms of these constructs and constraints.

For each equipment unit j we define a unary resource called *Unit[j]* and for each resource r (e.g. cooling water) we define a discrete resource *Utility[r]* with a maximum capacity R_r^{MAX}. Furthermore, for each state s we define a reservoir called *State[s]* with capacity C_s and initial level $S0_s$. For each binary Z_{ic} that is equal to 1 in the current optimal solution of the master problem (i.e. copy c of task i is carried out) we define an activity called *Task[i,c]* with duration D_{ic}. We also define a dummy activity *MS* with zero duration and no resource requirements. The reason we introduce *MS*, is because Constraint Programming is more efficient when the objective function is a function of one or few variables. Moreover, if a dummy activity is not used, the objective function for the minimization of makespan will be the minimization of the maximum finish time (i.e. a min max problem), which will require the introduction of additional constraints. If there are orders with due dates, and D is the set of orders for final products, *D(s)* is the set of orders for state s (i.e. $D=\cup_s D(s)$), and for each $d \in D$, AD_d is the amount due and TD_d is the due date, we also define $|D|$ activities, called *Order[d]*, with zero duration that are used for the "representation" of the orders. The corresponding declarations in OPL modeling language are:

UnaryResource Unit[j in Units]; (11)

DiscreteResource Utility[r in Utilities] (R_r^{MAX}); (12)

Reservoir State[s in States] (C_s, $S0_s$); (13)

Activity Task[i in Tasks, c in Copies] (D_{ic}); (14)

Activity MS (0); (15)

Activity Order[d in Orders] (0); (16)

The CP subproblem (SP) consists of equations (17) to (32):

$$B_i^{MIN} \le B_{ic} \le B_i^{MAX} \quad \forall i, \forall c \in C_i \tag{17}$$

$$D_{ic} = \alpha_i + \beta_i B_{ic} \quad \forall i \in ZW, \forall c \in C_i \tag{18a}$$

$$D_{ic} \ge \alpha_i + \beta_i B_{ic} \quad \forall i \notin ZW, \forall c \in C_i \tag{18b}$$

$$B_{ics}^I = \rho_{is}^I B_{ic} \quad \forall i, \forall c \in C_i, \forall s \tag{19}$$

$$B_{ics}^O = \rho_{is}^O B_{ic} \quad \forall i, \forall c \in C_i, \forall s \tag{20}$$

$$R_{icr} = \gamma_{ir} + \delta_{ir} B_{ic} \quad \forall i, \forall c \in C_i \tag{21}$$

$$\sum_i \sum_c B_{ics}^O \ge d_s \quad \forall s \in FP \tag{22}$$

Task[i,c] requires Unit[j] $\forall j, \forall i \in I(j), \forall c \in C_i$ (23)

Task[i,c] requires R_{icr} *Utility[r]* $\forall i, \forall c \in C_i, \forall r$ (24)

Task[i,c] consumes B^I_{ics} *State[s]*	$\forall i, \forall c \in C_i, \forall s$	(25)
Task[i,c] produces B^O_{ics} *State[s]*	$\forall i, \forall c \in C_i, \forall s$	(26)
Order[d].start $= TD_d$	$\forall d$	(27)
Order[d] consumes AD_d *State[s]*	$\forall s, \ \forall d \in D(s)$	(28)
Task[i,c].end $\leq$ *MS.start*	$\forall i, \forall c \in C_i$	(29)
Task[i,c] precedes Task[i,c+1]	$\forall i, \forall c < \lvert C_i \rvert$	(30)
Process Network Specific Constraints		(31)
(Optional) Objective Function		(32)

The batch size of activities is bounded by equation (17). If Zero Wait storage policy applies, the product of a task must be immediately transferred to the next task, and thus the duration of the task is exactly equal to its processing time as in (18a); for any other storage policy the material can be temporarily stored in the equipment unit, which means that the time during which equipment unit is "used" by *Task[i,c]* can be greater than the actual processing time [constraint (18b)]. The amount of reservoir *State[s]* consumed/produced by activity *Task[i,c]* is calculated by equation (19)/(20); the amount of discrete resource *Utility[r]* required by activity *Task[i,c]* is calculated in (21); the condition that the amount of final products should meet the demand is enforced by (22), where d_s is the total demand for state s. Parameter d_s is either given (in the case of fixed demand with no due dates) or calculated by,

$$d_s = \sum_{d \in D(s)} AD_d \quad \forall s$$

Special CP constructs and global constraints are used in equations (23) to (31). Constraint (23) enforces that all tasks in *I(j)* are assigned to unary resource *Unit[j]*. The consumption of R_{icr} units of discrete resource *Utility[r]* by activity *Task[i,c]* is enforced in (24), and the consumption/production of B^I_{ics} /B^O_{ics} units of reservoir *State[s]* by *Task[i,c]* is enforced by constraint (25)/(26). Orders with due dates are modeled through constraints (27) and (28). Each order is executed at its due time [constraint (27)], and the amount delivered is equal to the amount due [constraint (28)]. In (29) the end time of all activities is restricted to be smaller than the start of activity *MS*, and *MS* has, (a) fixed finish time equal to H when the objective is the maximization of profit over a fixed time horizon, or the minimization of cost for fixed demand and due dates, and (b) a variable finish time when the objective is the minimization of makespan for fixed demand with no due dates. Constraint (30) is a symmetry-breaking constraint that reduces the number of possible configurations by imposing a sequence between copies of the same task.

Constraints that describe some special features of the process network, or a special structure that can be exploited are included in (31). An example of such a feature is a Zero Wait state that is produced by only one task A and consumed by only one task B. In such a case, we can infer that every time task A takes place it must be immediately followed by task B, and that the batchsizes of consecutive batches of tasks A and B should be equal. Therefore, we can add the following constraints that greatly enhance the computational efficiency of the CP solver:

Task[A,c].end = Task[B,c].start	$\forall c$	(31a)
$BS_{Ac} = BS_{Bc}$	$\forall c$	(31b)

Depending on the nature of the problem (constant vs. variable processing times) and the objective function, we may want to solve the CP subproblem as one feasibility problem, as successive feasibility problems or as an optimization problem (details in section 4.6.2.). If the CP is an optimization problem we add the objective (32) whose exact form is given in section 4.5.

4.3. Time Domain Reduction

The performance of the proposed model depends on how fast we solve models (MP) and (SP), and the number of iterations needed to generate solutions and prove optimality. It is crucial, consequently, to exclude infeasible or suboptimal assignments of tasks as soon as possible. Preprocessing enhances the performance of the algorithm by (a) reducing the domains of certain variables, and (b) creating strong cuts that are added in the cut-pool of the master problem and are used to eliminate a priori a number of potential configurations.

4.3.1. Earliest Start Time and Latest Finish Time of Units

To illustrate the domain reduction for units, consider the example of Figure 4, where units U1, U2 and U3 are used for the production of products A, B and C, for which we assume that the duration of all tasks is 2 hours, the time horizon is 12 hours, and that at most two copies of the same task can take place (i.e. $|C_i|=2$ $\forall i$). Specifically, for unit U2, replacing durations from (3) into (2) we get:

$$2Z_{TA2,1}+2Z_{TA2,2}+2Z_{TB2,1}+2Z_{TB2,2}+2Z_{TC2,1}+2Z_{TC2,2} \leq 12$$

which implies that all 26 possible assignments are feasible since the constraint is satisfied if all Z_{ic} are equal to 1.

In the case where there are no intermediates SA1, SB1 and SC1 at $t=0$, however, no task can be carried out in unit U2 before $t=2$, i.e. before one of the states used as input in unit U2 is produced. Hence, the Earliest Start Time (EST) for any task in unit U2 is 2 hours. Similarly, since the time horizon is 12 hours, any amount of intermediates SA2, SB2 and SC2 produced after $t=10$ will not be used for the production of final products, which means that in all realistic solutions all tasks assigned to U2 must finish at or before $t=10$; i.e. the Latest Finish Time (LFT) of unit U2 is 10. When the time horizon is not fixed instead of using LFT we use the Shortest Tail (ST) which is the difference between the variable scheduling horizon and the LFT. In this example, the Shortest Tail (ST) of unit U2 is 2 hours.

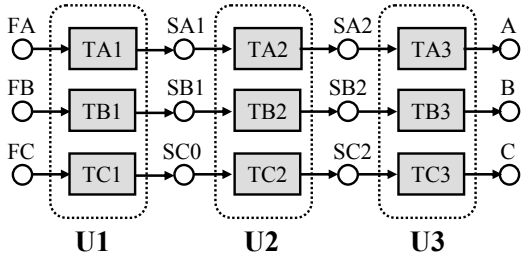


Figure 4: Flow diagram of Example.

Using this insight, we can rewrite the assignment constraint for unit U2 as follows:

$$2Z_{TA2,1}+2Z_{TA2,2}+2Z_{TB2,1}+2Z_{TB2,2}+2Z_{TC2,1}+2Z_{TC2,2} \leq LFT_{U2} - EST_{U2} = (H-ST_{U2}) - EST_{U2} = 8$$

which allows only up to four tasks assigned to unit U2, i.e. only 22 out of the total 26 assignments.
Note that similar tightening can be also performed when processing times are variables; in the example of Figure 4, we need only use the minimum, instead of the fixed, processing times of tasks TA1, TB1 and TC1 to calculate the EST of unit U2, and the minimum processing time of tasks TA3, TB3 and TC3 to calculate the ST of unit U3.
In the general case assignment constraint (2) is tightened as follows,

$$\sum_{i \in I(j)} \sum_{c} D_{ic} \leq (MS - ST_j) - EST_j \quad \forall j \tag{33}$$

where ST_j and EST_j are the Shortest Tail and Earliest Start Time of unit j, respectively, and MS is the time horizon (equal to H in the case of fixed time horizon). To calculate the EST and ST of unit j we need to calculate first the EST and ST of the tasks performed in j, which is described in the next section.

4.3.2. Earliest Start Time and Latest Finish Time of Tasks

To illustrate the domain reduction for tasks, consider the example of Figure 5. The duration of all tasks is 2 hours, the time horizon is 16 hours, each task can be carried out at most three times and unit U1 is used for tasks T1 and T8. If no intermediates are available at $t=0$, the EST and LTF for task T1 are 0 and 4 hours, respectively, and for task T8 EST is equal to 6 and LFT is equal to 10. This means that we can write the following constraints,

$$2Z_{T1,1}+2Z_{T1,2}+2Z_{T1,3} \leq 4 - 0 = 4 \qquad (34)$$

$$2Z_{T8,1}+2Z_{T8,}+2Z_{T8,3} \leq 10 - 6 = 4 \qquad (35)$$

Furthermore, the EST and LTF for unit U1 are $0=min\{EST_{T1},EST_{T8}\}$ and $10=max\{LFT_{T1},LFT_{T8}\}$, respectively, and the assignment constraint for unit U1 is,

$$(2Z_{T1,1} + 2Z_{T1,2}+2Z_{T1,3}) + (2Z_{T8,1}+2Z_{T8,2}+2Z_{T8,3}) \leq 10 - 0 = 10 \qquad (36)$$

Note that the LHS of constraint (36) is the sum of the LHSs of constraints (34) and (35), whereas the RHS of constraint (36) is larger than the sum of the sum of the RHSs of constraints (34) and (35), which implies that constraint (36) is a relaxation of constraints (34) and (35). In general, thus, we can get a tighter formulation by adding the constraints that restrict the sum of the processing times of the copies of the same task. Thus, for the general case, we add the following constraint for each task in $I(j)$,

$$\sum_c D_{ic} \leq (MS - ST_i) - EST_i \quad \forall i \in I(j) \qquad (37)$$

where ST_i and EST_i are the Shortest Tail and the Earliest Start Time of task i, respectively. We can therefore replace equation (2) by equations (33) and (37), and hence the master problem (MP) consists of equations (3) – (10), (33) and (37).

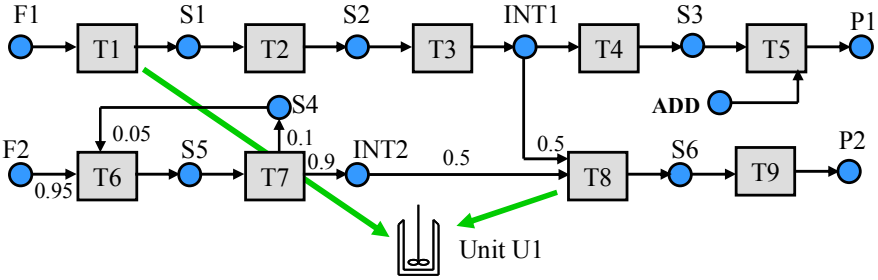


Figure 5: Process network of Example 2.

Note that for a general batch plant the calculations for EST and ST (or LFT) are not equivalent to a time window calculations for flow-shop batch plants. The main reasons for this are: (a) batch splitting and mixing is allowed, and (b) inventory for intermediate states may be available at $t=0$. In principle, EST and ST of tasks can be determined by inspection, but here we use a general, graph-theoretic algorithm that relies on identifying shortest and longest paths within the STN.

We define parameter AT_s as the time at which state s becomes available. States that are available at the beginning of the time horizon have $AT_s=0$. In order to calculate the EST of a task we need to know the AT_s of all the states that are consumed by task i. In order to calculate the AT of a state we need to know the EST and the (minimum) processing time of all tasks producing this state. If $SI(i)$ is the set of states consumed by task

i, $O(s)$ is the set of tasks that produce state s, DT_s is the earliest delivery time of state s (zero if state s is available at $t=0$), and D_i^{MIN} is the minimum duration of task i, the procedure, in summary, is as follows:

The EST of task i is calculated by: $EST_i = max_{s \in SI(i)} \{AT_s\}$
The AT of state s is calculated by: $AT_s = min\{DT_s, min_{i \in O(s)} \{EST_i + D_i^{MIN}\}\}$

For the calculation of the Shortest Tail ST_i of task i we follow similar rules but we start from the final products ($s \in FP$), and for each state $s \in S$ we calculate the minimum time MT_s needed for state s to be used for the production of a final product. Is $SO(s)$ is the set of states produced by task i, and $I(s)$ is the set of tasks consuming state s, the summary of the backward procedure for the calculation of the ST of tasks (if there are no recycle streams) is the following:

The ST of task i is calculated by: $ST_i = min_{s \in SO(i)} \{MT_s\}$
The MT of state s is calculated by: $MT_s = min_{i \in I(s)} \{ST_i + D_i^{MIN}\}$

If ET (UT) is the set of examined (unexamined) tasks, and ES (US) is the set of examined (unexamined) states, and D_i^{MIN} is the minimum (or fixed) processing time of task i, a straightforward computer implementation of the procedure for networks with variable or constant processing times and no recycle streams is the following:

Calculation of EST:

Initialization: $ET = \varnothing$, $UT = I$, $ES = \{s \in S |\ S0_s > 0\}$, $US = S \backslash ES$, $AT_s = 0\ \forall s \in ES$
Until $UT = \varnothing$ do
  For all $i \in UT$
    If $ES \cap SI(i) = SI(i)$ then $EST(i) = max_{s \in SI(i)} \{AT_s\}$
      $ET = ET \cup \{i\}$
      $UT = UT \backslash \{i\}$
  For all $s \in US$
    If $ET \cap O(s) = O(s)$ then $AT(s) = min\{DT_s, min_{i \in O(s)} \{EST_i + D_i^{MIN}\}\}$
      $ES = ES \cup \{s\}$
      $US = US \backslash \{s\}$

Calculation of ST:

Initialization: $ET = \varnothing$, $UT = I$, $ES = FP$, $US = S \backslash FP$, $MT_s = 0\ \forall s \in ES$
Until $UT = \varnothing$ do
  For all $i \in UT$
    If $ES \cap SO(i) = SO(i)$ then $ST(i) = min_{s \in SO(i)} \{MT_s\}$
      $ET = ET \cup \{i\}$
      $UT = UT \backslash \{i\}$
  For all $s \in US$
    If $ET \cap I(s) = I(s)$ then $MT(s) = min_{i \in I(s)} \{ST_i + D_i^{MIN}\}$
      $ES = ES \cup \{s\}$
      $US = US \backslash \{s\}$

An example of its application is presented in Appendix D. In case there are recycle streams and RC is the set of recycled states, the calculation is more complicated. We initially exclude states in RC (i.e. $US = S \backslash (FP \cup RC)$, calculate EST and ST for all states and then re-calculate taking into account the recycled states. Having calculated the EST and ST of all tasks, the EST and ST of units are easily calculated by,

$$EST_j = min\{EST_i \,|\, i \in I(j)\} \tag{38}$$

$$ST_j = min\{ST_i \,|\, i \in I(j)\} \tag{39}$$

4.4. Integer Cuts

The preprocessing described in the previous section reduces the number of potential configurations by tightening the existing and adding new assignment constraints in the MILP master problem (MP). Another way to reduce the number of potential assignments is by generating integer cuts that explicitly forbid some infeasible or suboptimal configurations that cannot be excluded by the tightening of domains. While one simple integer cut is added at each iteration of the proposed algorithm, some integer cuts can be inferred during preprocessing. The simple integer cuts and two more classes of cuts derived during preprocessing are discussed next.

4.4.1. Integer Cuts I

The first class of integer cuts comprises of the simplest and weakest cuts that exclude the current assignment, and its general form is:

$$\sum_{(i,c)\in B^k} Z_{ic} + \sum_{(i,c)\in N^k} (1 - Z_{ic}) \le \left|B^k\right| + \left|N^k\right| - 1 \quad \text{or} \quad \sum_{(i,c)\in B^k} Z_{ic} - \sum_{(i,c)\in N^k} Z_{ic} \le \left|B^k\right| - 1 \tag{40}$$

where B^k and N^k are the sets of (i,c) pairs for which $Z_{ic} = 1$ and $Z_{ic}=0$, respectively, in the current iteration k. The integer cut in (40) excludes only the current assignment. A stronger cut is one that excludes the current assignment k and any other assignment that is a superset of assignment k; i.e. any assignment l for which $B^l \supseteq B^k$. The general form of this cut is:

$$\sum_{(i,c)\in B^k} Z_{ic} \le \left|B^k\right| - 1 \tag{41}$$

While stronger, the integer cuts in (41) may cutoff feasible solutions. If economy of scale holds for batch-sizes and processing times, then they can be safely used when the time horizon is fixed and the current assignment B^k is infeasible, since any assignment B^l with $B^l \supseteq B^k$ will also be infeasible. But they cannot be used if the current assignment is feasible, because an assignment B^l with $B^l \supseteq B^k$ may yield a higher profit. A case where they can be added in every iteration is when the objective is the minimization of makespan and processing times are constant; in this case, if an assignment B^k satisfies the demand and yields a makespan MS_k, then any assignment l that includes more tasks would yield a makespan $MK_l \geq MK_k$.

4.4.2. Integer Cuts II

The motivation behind these cuts is to decompose the STN into subnetworks in order to identify infeasible assignments of tasks as early as possible. To illustrate the derivation of this class of integer cuts consider the process network shown in Figure 6. There, raw material RM1 is converted into intermediate INT1 (via tasks T11, T12, T13 or T14 that are performed in unit U1), raw material RM2 is converted into intermediate INT2 (via tasks T21 or T22 that are performed in unit U2), intermediates INT1 and INT2 are fed in a 4:1 ratio for task T31 in unit U1 to produce final product P1, and intermediate INT2 is converted into final product P2 (through task T32 also in unit U3). Each task has a constant cost c and a constant processing time pt shown in Figure 6. The capacity of units U1, U2 and U3 is 5, 10 and 10 kg respectively. There is one order of 5 kg of P1 to be met at $t=6$ and one order of 5 kg of P2 to be met at $t=4$. The objective is to find a schedule of minimum cost that satisfies the demand and the due dates.

The Earliest Start Time (EST) for all tasks performed in units U1 and U2 is 0, while for tasks T31 and T32 it is 2. Since the latest due date is at $t=6$, the scheduling horizon is 6 hours the Latest Finish Time (LFT) of all tasks performed in units U1 and U2 is 4 and for T31and T32 in unit U3 is 6 (or $ST_{U1}=ST_{U2}=2$ and $ST_{U3}=0$). Using these values, the proposed iterative scheme yields the series of assignments shown in Figure 7 (tasks that are carried out are highlighted). The first three assignments include one of the tasks performed in U1, and tasks T22, T31 and T32 and they are all infeasible. The reason for the infeasibility is that the duration of task T22 is 3 hours, which means that intermediate INT2 becomes available at $t=3$, and thus the order of P2

cannot be met on time. We can easily infer that in any feasible solution, task T21 must be carried out because this is the only way intermediate INT2 becomes available at $t=2$. Hence, if we had this insight before starting the iterative scheme and we had fixed binary $Z_{T21,1}$ to 1, we would get the 4th assignment at the first iteration. Note that similar reasoning can also be applied when processing times are variables, using the minimum instead of the fixed processing time of tasks.

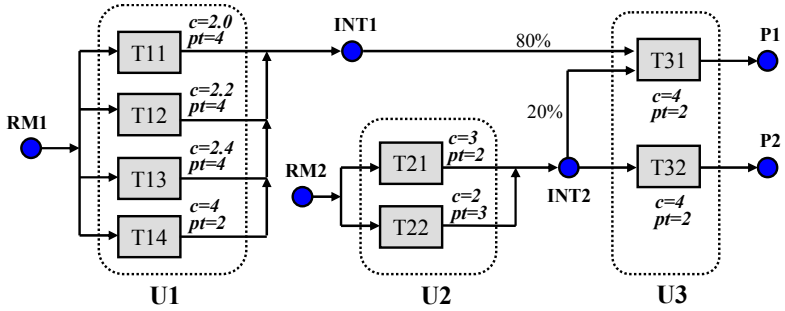
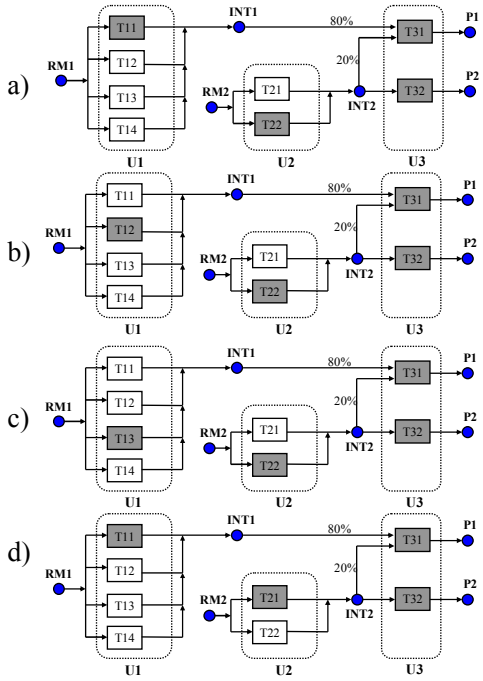


Figure 6: Flow diagram for integer cuts.



Iteration 1:
Cost = 12.0
INFEASIBLE ASSIGNMENT
Cut: $Z_{T11,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1}\leq 3$

Iteration 2:
Cost = 12.2
INFEASIBLE ASSIGNMENT
Cut: $Z_{T12,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1}\leq 3$

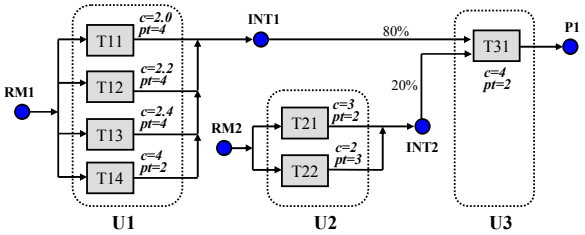
Iteration 3:
Cost = 12.4
INFEASIBLE ASSIGNMENT
Cut: $Z_{T13,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1}\leq 3$

Iteration 4:
Cost = 13.0
FEASIBLE ASSIGNMENT

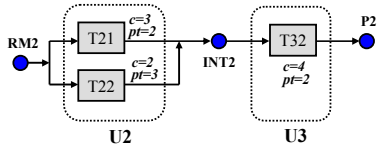
Figure 7: Assignments of Master Problem

Based on this simple example we can in general decompose the process network into smaller subnetworks and try to infer general rules (fixed assignments or integer cuts) that hold true in any solution. By subnetworks we mean a subset of tasks that, (a) can be easily identified (e.g. with a procedure similar to the one described in the next paragraph), and (b) are likely to give useful information (i.e. information that holds true for larger sets of tasks and ideally for the entire process network). Such a subnetwork can be, for example, the set of tasks used for the production of one individual product. In the example of Figure 6 the process network can be decomposed into the two subnetworks shown in Figure 8. While no inference can be made by the subnetwork for the production of P1, the second subnetwork reveals that in order to meet the due date for product P2 we need to carry out task T21 (i.e. $Z_{T21,1}=1$ in every feasible solution). Having fixed $Z_{T21,1}$, we then obtain the optimal assignment (Figure 7d) at the first iteration. Since (a) the capacity of unit U2 is 10 kg, (b) the processing time of task T21 does not depend on the batch size and (c) for both orders we

need 6 kg of INT2, we could have further inferred that no other task is needed to be carried out in unit U2 (i.e. $Z_{T21,c} = Z_{T22,c} = 0, \ \forall c$).



(a) Subnetwork for the production of P1



(b) Subnetwork for the production of P2

Figure 8: Subnetworks of process network of Figure 6.

In general, any network can be decomposed in $|FP|$ subnetworks, and each subnetwork SN consists of a subset of tasks $SNI \subset I$ and a subset of states $SNS \subset S$. To identify such a subnetwork, we start from a final product $s^* \in FP$ and, backtracking, we add to SNI and SNS all the tasks and states that are involved in the production of s^*. If $O(s)$ is the set of tasks that produce state s, $SI(i)$ is the set of input states to task i, and RM is the set of raw material states, the formal procedure for the identification of the subnetwork that corresponds to final product s^* is the following:

SUBNET_PROC:

Initialization: $SNI = \varnothing$, $SNS = \{s^*\}$, $CI = O(s^*)$
Until $CI = \varnothing$
  For all $i \in CI$
    $SNI = SNI \cup \{i\}$, $CI = CI \setminus \{i\}$
    For all $s \in SI(i)$
      $SNS = SNS \cup \{s\}$, $CI = CI \cup O(s)$

For each subnetwork identified by SUBNET_PROC, we apply the proposed iterative algorithm and derive cuts that have the general form of constraint (40) and exclude configurations that are infeasible for the specific subnetwork; i.e. Integer Cuts II have the form of (40), where $i \in SNI \cap B^k$. Moreover, since Integer Cuts II have the form of equation (40), they exclude only one assignment and, thus, can be used for both constant and variable processing times. Network-specific constraints that include only the Z_{ic} binaries, similar to constraints in (31) of the CP subproblem, can be added also in the master problem and grouped as Integer Cuts II. For the case of Zero Wait state that is produced by only one task A and consumed by only one task B, for instance, we can add the following equation that allows assignments that consist of the same number of copies of tasks A and B:

$$\sum_c Z_{Ac} = \sum_c Z_{Bc} \qquad (42)$$

Note, that the restriction for zero-wait storage policy for one or more states makes, in general, continuous MILP models very difficult to solve due to the additional time points needed for the finish time of the tasks that produce ZW states. In contrast, this restriction makes the CP subproblem of the proposed approach easier to solve because it becomes more tightly constrained.

Depending on the plant topology, other sub-networks can also be studied. The equations added in this step (fixing of variables or integer cuts) are Integer Cuts II. Any other cut that can be extracted from the structure of the process network (based on heuristic rules, prior knowledge of the process network or any other source) can be grouped in this class.

4.4.3. Integer Cuts III

In batch plants it is very common to have a number of identical parallel units in one stage, or *similar* tasks that are carried out in the same unit. The *similarity* of these tasks is that they have the same processing time but slightly different utility requirements or cost. Tasks T11, T12 and T13 of the network in Figure 6, for example, are similar because they have equal processing times but different costs. This similarity usually implies that if one of these tasks leads to an infeasible assignment, every other similar task will also give an infeasible assignment. The high frequency at which these configurations appear in many problems led us to develop a new class of integer cuts. The aim is to exclude similar assignments in as few iterations as possible.

To motivate this class of cuts, consider the previous process network (Figure 6), with two orders of 5 kg each for P1 due at *t=4* and *t=10*, and one order of 5 kg of P2 due at *t=8*. The domain reduction for tasks and units yields the results of Table 1.

Table 1: Earliest start and latest finish times of Example.

	T11	T12	T13	T14	T21	T22	T31	T32	U1	U2	U3
EST	0	0	0	0	0	0	2	2	0	0	2
LFT	8	8	8	8	8	8	10	8	8	8	10

Assuming that the time horizon of the two subnetworks used for the production of P1 and P2 is 10 and 8 hours, respectively, the study of the two subnetworks does not reveal any additional information (i.e. we cannot derive any integer cut of type II). Thus, the iterative scheme yields the assignments that are shown in Figure 9, and the optimal solution is found in the 8th iteration. Note that assignments 1, 2 and 3 are practically the same because the processing times of tasks T11, T12 and T13 are the same. Similarly, assignments 4, 5 and 6 are identical to each other in terms of processing times. To exclude these two groups of similar assignments, however, we need the first six integer cuts shown in Figure 9. It would be very useful, hence, to develop a class of integer cuts that excludes a whole group of similar assignments. To do so, we need to identify the classes of similar equipment units and define new binary variables.

The general procedure for deriving the third class of integer cuts is the following:

(a) Identify classes of *similar* tasks; i.e. tasks that perform identical operations (same input and output states and same conversion factors) and with equal processing times. The set of similar tasks that belong to class *g* is denoted by *I(g)*.

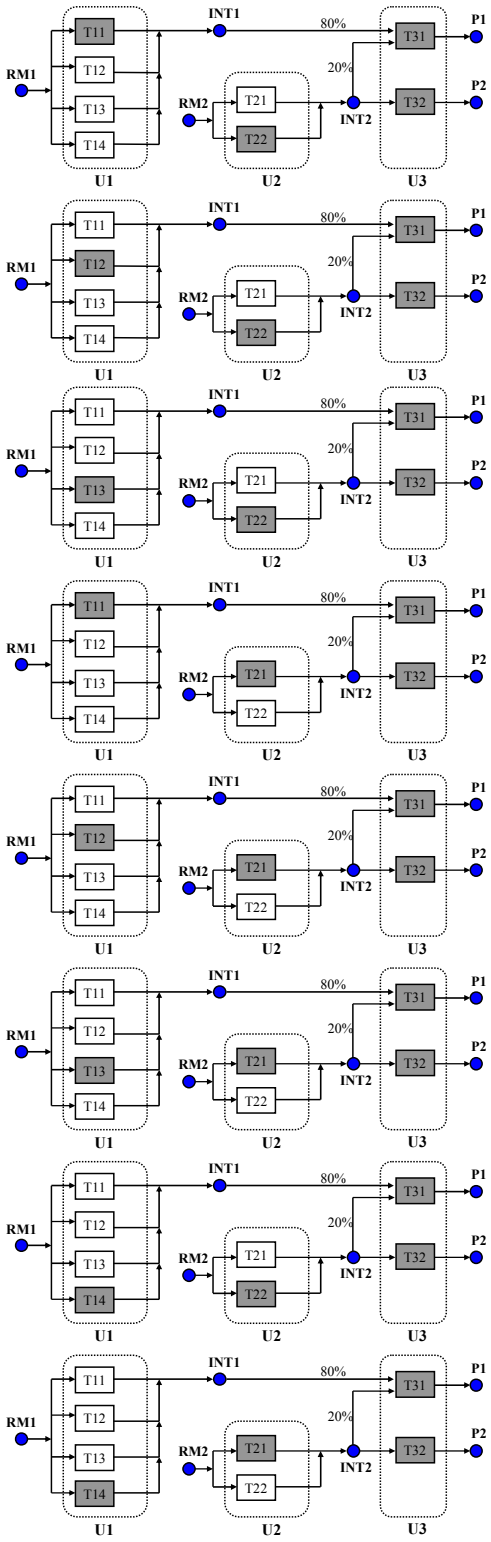
(b) Define new binary variables Y_{gcn} for each class *g* of similar tasks; binary Y_{gcn} is 1 if, in the current assignment, there are n of the c^{th} copies of tasks in *I(g)*. The value of the Y_{gcn} binaries is determined by the following equations that link the *class* binaries Y_{gcn} to the *task* binaries Z_{ic},

$$\sum_{i \in I(g)} Z_{ic} = \sum_{n=1..|I(g)|} n \cdot Y_{gcn} \quad \forall g, \forall c \tag{43}$$

$$\sum_{n=1..|I(g)|} Y_{gcn} = 1 \quad \forall g, \forall c \tag{44}$$

In the example of Figure 6, for instance, if *A* is the class of tasks {T11, T12, T13} and in the current assignment there are two copies of task T11 ($Z_{T11,c}$ *=1*, for *c=1, 2*), and one copy of task T12 (Z_{T12c} *=1,* for *c=1*), we will have Y_{A12} *=1*, Y_{A21} *=1*.

(c) Develop integer cuts in the mixed (Y_{gcn}, Z_{ic}) space: for any assignment that includes a task $i \in I(g)$ use equations (43) and (44) to calculate binaries Y_{gcn} and develop a new cut (cut III) by replacing binaries Z_{ic} with binary Y_{gnc} in the corresponding integer cut I. Any integer cut that includes Y_{gcn} binaries is called an Integer Cut III.



Iteration 1:
Cost = 12.0
INFEASIBLE ASSIGNMENT
Cut: $Z_{T11,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1} \leq 3$

Iteration 2:
Cost = 12.2
INFEASIBLE ASSIGNMENT
Cut: $Z_{T12,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1} \leq 3$

Iteration 3:
Cost = 12.4
INFEASIBLE ASSIGNMENT
Cut: $Z_{T13,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1} \leq 3$

Iteration 4:
Cost = 13.0
INFEASIBLE ASSIGNMENT
Cut: $Z_{T11,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1} \leq 3$

Iteration 5:
Cost = 13.2
INFEASIBLE ASSIGNMENT
Cut: $Z_{T12,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1} \leq 3$

Iteration 6:
Cost = 13.4
INFEASIBLE ASSIGNMENT
Cut: $Z_{T13,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1} \leq 3$

Iteration 7:
Cost = 14.0
INFEASIBLE ASSIGNMENT
Cut: $Z_{T14,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1} \leq 3$

Iteration 8:
Cost = 15.0
FEASIBLE ASSIGNMENT

Figure 9: Assignments derived by the master problem.

In the example of Figure 6, tasks T11, T12 and T13 make up a class of *similar* tasks. Let A be this class. The seven first assignments of Figure 9 and the corresponding integer cuts are given in the second and third column, respectively, of Table 2.

Table 2: Assignments of Figure 9 and integer cuts I and III.

Iter.	Tasks of assignment	Integer Cut I	Equation (43)	Integer Cut III
1st	*T11, T22, T31, T32*	$Z_{T11,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1}\leq 3$	$Z_{T11,1}=1\rightarrow Y_{A,1,1}=1$	$Y_{A,1,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1}\leq 3$
2nd	*T12, T22, T31, T32*	$Z_{T12,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1}\leq 3$	$Z_{T12,1}=1\rightarrow Y_{A,1,1}=1$	
3rd	*T13, T22, T31, T32*	$Z_{T13,1}+Z_{T22,1}+Z_{T31,1}+Z_{T32,1}\leq 3$	$Z_{T13,1}=1\rightarrow Y_{A,1,1}=1$	
4th	*T11, T21, T31, T32*	$Z_{T11,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1}\leq 3$	$Z_{T11,1}=1\rightarrow Y_{A,1,1}=1$	$Y_{A,1,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1}\leq 3$
5th	*T12, T21, T31, T32*	$Z_{T12,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1}\leq 3$	$Z_{T12,1}=1\rightarrow Y_{A,1,1}=1$	
6th	*T13, T21, T31, T32*	$Z_{T13,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1}\leq 3$	$Z_{T13,1}=1\rightarrow Y_{A,1,1}=1$	
7th	*T14, T21, T31, T32*	$Z_{T14,1}+Z_{T21,1}+Z_{T31,1}+Z_{T32,1}\leq 3$		

Using equations (43) and (44) we can calculate the value of binary Y_{Anc} at each iteration (shown in the fourth column) and develop the new integer cuts III by replacing Z_{ic} for tasks in A by binary Y_{Acn} (fifth column). Note that the six first assignments correspond to only two different assignments in the (Z_{ic}, Y_{gcn}) – space. The first integer cut III excludes the first three assignments and the second cut excludes the next three assignments. Thus, we could have found the optimal assignment in only four iterations if we had added the two integer cuts of the fifth column. Integer Cuts III can be used for both constant and variable processing times.

4.5. Objective Functions and Properties of Upper and Lower Bounds

4.5.1. Maximization of Profit over a Fixed Time Horizon

Master problem objective: $\max \sum_{s\in FP} \zeta_s \hat{S}_s$ (10a)

Subproblem objective: $\max \sum_{s\in FP} \sum_{i\in I(s)} \sum_{c\in C_i} \zeta_s B^O_{ics}$ (32a)

where ζ_s is the price of final product $s\in FP$, *MS* in constraints (33) and (36), is equal to the fixed time horizon *H*, and the dummy activity *MS* in the CP subproblem is fixed to start at $t=H$. The maximization of production is a special case of this objective with $\zeta_s=1$ for all *s*.

Since the objective is to maximize profit, assignments that are feasible with regard to each unit separately are obtained by the master problem, and the upper bound provided by the master problem is often very large. Even feasible assignments yield upper bounds that are usually much higher than their actual objective value (calculated by the subproblem). The first CP subproblems are usually infeasible and the optimal solution usually corresponds to the solution of one of the first feasible assignments, but a significant number of additional iterations may be needed to prove optimality. A typical graph of the upper and lower bounds is given in Figure 10a, where some assignments are infeasible, the upper-lower bound gap is large, and many additional iterations are needed to prove optimality.

4.5.2. Minimization of Makespan for Fixed Demand (with no Due Dates)

Master problem objective: min *MS* (10b)

Subproblem objective: min *MS.end* (32b)

where *MS* in constraints (33) and (34) is a variable, and *MS.end* is the finishing time of the dummy task *MS* in the CP subproblem.

Any assignment that fulfills the fixed demand is feasible for the CP subproblem, but the lower bound on makespan calculated by the master problem is smaller than the actual makespan calculated by the subproblem. Thus, the optimal assignment is usually found early in the iterative process and it takes a few more iterations to prove optimality. A typical graph of the upper and lower bound for the minimization of makespan problem is shown in Figure 10b, where all assignments are feasible, the upper-lower bound gap is small and few additional iterations are needed to prove optimality.

4.5.3. Minimization of Production Cost for Fixed Demand with Due Dates

Master problem objective: $$\min \sum_{i} \sum_{c \in C_i} PC_i Z_{ic} \qquad (10c)$$

Subproblem is a feasibility problem
where PC_i is the constant cost of carrying out task i.

The production cost of an assignment provided by the master problem is equal to the actual cost of this assignment, if feasible. Thus, the first feasible assignment is the optimal assignment, and the last iteration provides the only upper bound that is equal to the optimal solution and the current lower bound. This is illustrated in Figure 10c.

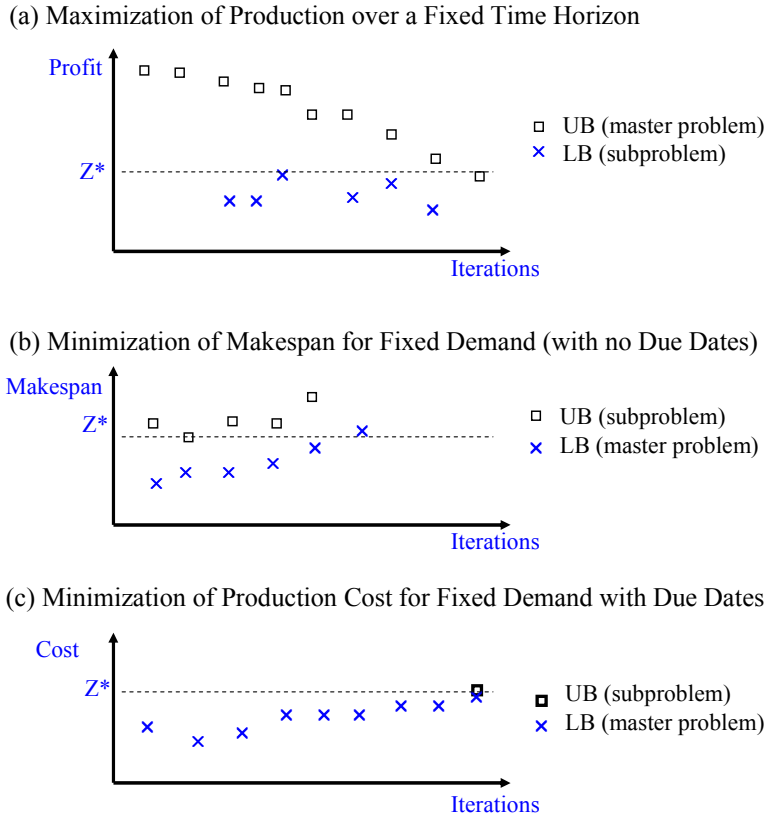


Figure 10: Graphs of upper and lower bounds for the three objectives.

Overall, the proposed method is computationally highly efficient when the objective function is the minimization of makespan or the minimization of assignment cost, while it is moderately efficient when the objective is the maximization of production or profit. The computational effort required for each iteration in the presence of intermediate due dates remains the same, but the number of iterations needed to find the optimal solution and prove optimality increases. Jain and Grossmann (2001) and Harjunkoski and

Grossmann (2002) proposed similar, though simpler, approaches for the minimization of assignment cost in single- and multi-stage batch plants, respectively, and showed that hybrid MIP/CP schemes are very effective.

4.6. Remarks

4.6.1. Validity of Integer Cuts

Integer Cuts I in (40) are always valid while Integer Cuts I in (41) are valid only in special cases (e.g. minimization of makespan with fixed processing times). In order to guarantee optimality, we use the form in (41) only when they do not cut off feasible solutions. Integer Cuts II can be (a) equations that describe some special structure or characteristic of the process network (e.g. constraint (42) for a ZW storage policy), or (b) integer cuts derived from the study of the subnetworks derived by the SUBNET_PROC procedure. In both cases they are always valid. Integer Cuts III are cuts that include the Y_{gcn} binaries, and they are very useful for multistage plants (Pinto and Grossmann, 1995) where in each stage there are many similar parallel units, and many orders with different due dates have to be processed. Note that these cuts are valid if added during preprocessing, and specifically, if added for the subnetworks identified by the SUBNET_PROC procedure, but they may cut off feasible solutions if added in the main iterative scheme. In the current implementation, however, they are added only for the subnetworks derived by the SUBNET_PROC procedure during the preprocessing phase and are always valid.

4.6.2. Convergence

Since there is a finite number of assignments, the proposed hybrid scheme of Figure 3 is guaranteed to converge in a finite number of iterations. Furthermore, the method is guaranteed to find the optimal solution, since the master problem is a relaxation, and the integer cuts we use are always valid as discussed in 4.6.1. The computational effort required for the convergence, however, varies considerably. Our computational studies show that, besides the size of the model, there are three main factors affecting the performance of the algorithm:

(a) The number of different possible assignments. Problems with many *similar* assignments are hard to solve because there are many assignments with similar objective function values and it is hard to prove optimality. Networks with many identical tasks or tasks that can be performed in many units usually belong to this group of problems, and require a large number of iterations.
(b) The objective function. The quality of the lower and upper bound depends heavily on the objective function. Specifically, since the master problem is a relaxation of the original problem, the tightness of the bound of the master problem is not always good. If the objective function of the original problem can be expressed in terms of the variables of the master problem the quality is good; otherwise it is not, and many iterations may be needed to close the gap.
(c) The nature of the processing times (constant vs. variable) and the mass fractions. Most of the CP global constraints accept integer arguments, which means that in the case of variable processing times or mass fractions with many significant digits, fine discretization is needed for variables B_{ic}, B^I_{ics}, B^O_{ics} and D_{ic} which are interconnected. In the other extreme, if processing times are constant and mass fractions are equal to 1, D_{ic} is constant and for the remaining variables a coarse discretization suffices.

The first two factors affect the number of iterations, while the third one affects the computational effort needed for the solution of the CP subproblem and is discussed in section 4.6.4. The domain reduction through preprocessing and the addition of integer cuts of type II and III is very successful in reducing the number of potential assignments with similar objective function values. Any further cuts, heuristic rules or other network-specific information that reduce the number of potential assignments can also be used to reduce the number of iterations.

4.6.3. Solution of Subproblem

The CP subproblem is a feasibility problem when we are trying to minimize cost and an optimization problem when we are trying to maximize profit or minimize makespan. For the minimization of cost the implementation is straightforward: the MILP master problem gives a non-decreasing lower bound and the CP subproblem checks feasibility. The first feasible subproblem gives an upper bound equal to the current lower bound, i.e. the optimal solution. When the CP subproblem is an optimization problem we can solve it either as an optimization problem, or as successive feasibility problems in which the bound on the objective function is updated. If solved as one optimization problem the implementation is straightforward, similar to the one for the minimization of cost, but it is usually very expensive to optimize a CP model. If it is solved as successive feasibility problems, the implementation is more complicated because it involves the solution of successive CP models but it is usually more efficient, since each successive tree search is more tightly constrained. When the objective is the minimization of makespan, the gap is small and thus few subproblems need to be solved in each major iteration. Hence, in this work, we always solve successive feasibility CP subproblems when we minimize makespan for fixed demand and in each subproblem we add the following constraint,

$$MS.end \leq MS^k$$

where MS^k is the makespan in the feasibility subproblem of the k^{th} iteration.
For the maximization of profit, the gap between the upper and the lower bound is larger and we use both approaches. The complete iterative schemes for all cases are shown in Figure 11, where for the maximization of profit we solve one optimization subproblem.

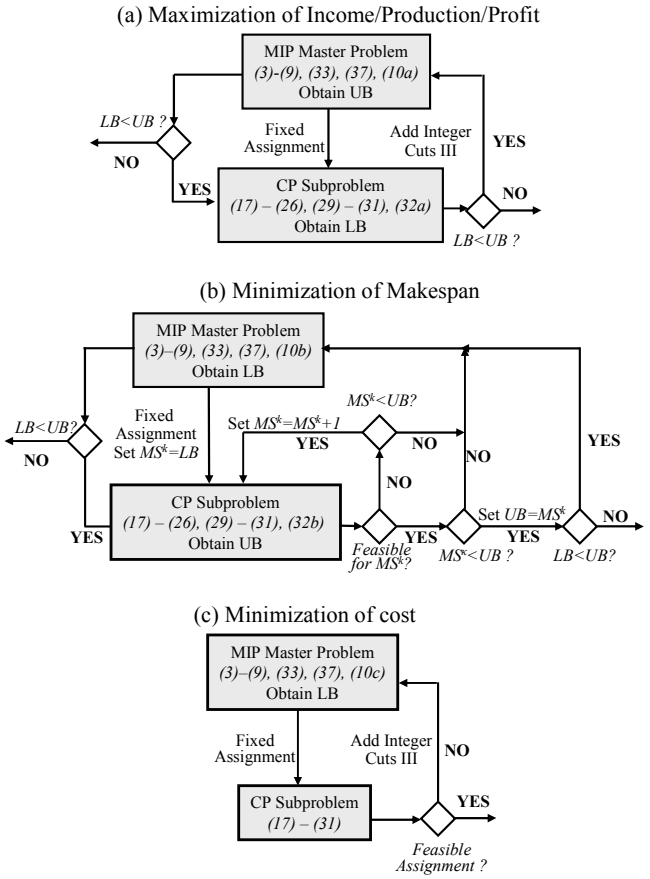


Figure 11: Iterative Scheme

4.6.4. Discretization

The global constraints *requires*, *consumes* and *produces* and the constructs *activity*, *discrete resource* and *reservoir* accept integer arguments; i.e. the arguments R_{icr}, B^I_{ics}, B^O_{ics} and D_{ic} in constraints (12), (14) and (24)-(26) must be integer variables, and the arguments R_r^{MAX}, C_s, and $S0_s$ in constraints (12) and (13) must be integer constants. Thus, variables D_{ic}, B_{ic}, R_{icr}, B^I_{ics} and B^O_{ics} of the CP subproblem must be integers, while they satisfy constraints (18a) – (21) that interconnect them. To ensure that these variables are integers we should discretize the domains of the variables so that α_i and γ_i are integers and the products of B_{ic} with β_i, δ_{ir}, ρ^I_{is} and ρ^O_{is} are also integers. Depending on the values of constants α_i, β_i, γ_{ir}, δ_{ir}, ρ^I_{is} and ρ^O_{is}, and the required resolution this may result in very large domains for variables D_{ic}, B_{ic}, R_{icr}, B^I_{ics} and B^O_{ics}. A general rule is that fine discretization is needed when the greatest common factor of α_i, β_i, γ_{ir}, δ_{ir}, ρ^I_{is} and ρ^O_{is} constants is very small.

Consider, for instance, that the processing time of task A is given by,

$D_A = 1 + 0.5\, B_A$ (in hours)

where B_A is the batch-size of task A in tons. Assuming that B_A takes values in [2,6], we can deduce that D_A takes values in [2,4]. In Constraint Programming, variables B_A and D_A must be integers, which means that B_A takes values in {2,3,4,5,6}. Since D_A should be an integer, we should scale our variables, i.e. the duration of the time intervals should be ½ hour instead of 1 hour and the processing time is given by:

$D_A = 2 + B_A$ (in ½ hours)

If finer resolution is required, we can also scale B_A variable, i.e. express B_A in hundreds of kilograms, which means that $B_A \in \{20, 21, \ldots 60\}$. In this case the time discretization should be in intervals of 3 minutes, i.e. the domain of D_A is $\{20, 21, 22, ..80\}$ and D_A is given by:

$D_A = 20 + B_A$ (in 3 mins)

In general, the finer the resolution the harder to solve a CP problem, but note that in CP we can use much finer resolution than in discrete-time STN models. Moreover, in CP we can approximate the values of some of these constants (as in discrete-time STN), but we can also approximate (overestimate) the processing time of tasks without rounding the batch size of a task. A common approximation, for example, is to replace constraint (18a) with the following two constraints that do not require that the RHS of (18a) is an integer:

$$D_{ic} \geq \alpha_i + \beta_i B_{ic} \quad \forall i \in ZW, \forall c \in C_i \qquad (18a')$$

$$D_{ic} < \alpha_i + \beta_i B_{ic} + 1 \quad \forall i \in ZW, \forall c \in C_i \qquad (18a'')$$

The duration of task A when B_A is expressed in tons (i.e. $B_A \in \{2,3,4,5,6\}$), for example, can be defined by its domain (i.e. $D_A \in \{2,3,4\}$) and the following two constraints (note that D_A is exact when B_A is even, and it is an overestimation of the actual processing time when B_A is odd):

$D_A \geq 1 + 0.5\, B_A$ (in hours)

$D_A < 1 + 0.5\, B_A + 1$ (in hours)

It is important to note that despite the approximations that might be needed, the proposed method yields consistently better solutions than the *exact* MILP models. As shown in the next paragraph, exact MILP models are computationally expensive to solve and thus yield suboptimal solutions, whereas the proposed method, in the worst case, yields an "approximation" of the optimal solution, which is better than an exact suboptimal solution.

5. Examples

To illustrate the implementation of the proposed algorithm we first solve two examples (from Papageorgiou and Pantelides, 1996) of multipurpose batch plants, assuming constant processing times and no resource constraints other than equipment. Computational results for constant and variable processing times and both maximization of profit and minimization of makespan for the first example are reported in section 6, as well as comparisons with the MILP model of Maravelias and Grossmann (2003a).

5.1. Example 1

A multi-stage batch plant is used for the production of three products P1, P2 and P3 as shown in Figure 12. Unlimited storage is available for raw materials and final products, finite intermediate storage is available for the intermediates produced in the first stage and zero-wait policy applies for the intermediates produced in the second stage. The objective is to maximize the overall production of the plant over a time horizon of 15 hours. Sufficient amounts of raw materials are available at the beginning of the scheduling horizon, while there are zero amounts of all other states. Unit and storage tank capacity data as well as processing time data are given in Appendix E.

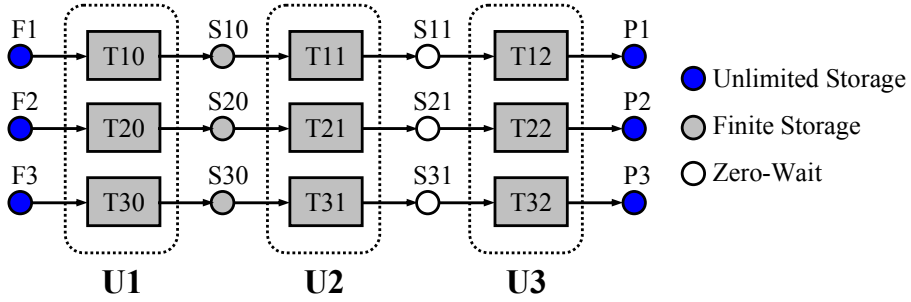


Figure 12: Process network of Example 1.

The EST and LFT determined in the preprocessing stage are reported in Table 3. No integer cuts of type II and III are derived due to the small size of the problem and the absence of similar tasks and units. If we allow up to five copies for all tasks, the optimal solution with an objective function value of 12 units is found in 4 iterations. The bounds and the assignments of all iterations are reported in Table 4, with the number of copies in parentheses if greater than 1. If a feasible solution has been found (i.e. a lower bound is available), the CP subproblem is solved as a feasibility subproblem where we are looking for solutions better than the current lower bound. The upper bound calculated by the master problem is equal to 12 for all iterations. The first assignment is infeasible. The second is feasible and yields a lower bound of 11. The third assignment is also infeasible. The fourth assignment is feasible and yields a solution of 12, which means that a lower bound of 12 is found, and since the upper bound is also 12, this is the optimal solution. The total CPU time is 0.48 seconds.

Table 3: Task EST and LFT for Example 1 (in hours)

Task/Unit	T10	T20	T30	T11	T21	T31	T12	T22	T32	U1	U2	U3
EST	0	0	0	4	3	2	6	5	3	0	2	3
LFT	4	4	3	2	2	2	0	0	0	3	2	0

Note that if the maximum number of copies were four rather than five, the optimal assignment is obtained and the optimality is proved in only one iteration. The Gantt chart of tasks for the optimal solution obtained is shown in Figure 13, where the batch size of tasks T10, T20 and T30 is 4 tons and the batch size of all other tasks is 2 tons. Note that the tasks in the third stage start as soon as the tasks in the second stage finish, due to the zero-wait policy for intermediates S11, S21 and S31.

Table 4: Progress of algorithm for Example 1.

Iteration	Master MILP Problem		CP Subproblem	
	UB	Assignment: Task (no of copies)	Solution	LB
1	12	T10, T11(2), T12(2), T20(2), T21(3), T22(3), T30, T31, T32	Infeasible	-
2	12	T10(2), T11(3), T12(3), T30(2), T31(3), T32(3)	Feasible with Z = 11	11
3	12	T10(2), T11(4), T12(4), T30(2), T31(2), T32(2)	Infeasible for Z > 11	11
4	**12**	T10, T11(2), T12(2), T20, T21(2), T22(2), T30, T31(2), T32(2)	Feasible for Z > 11	**12**

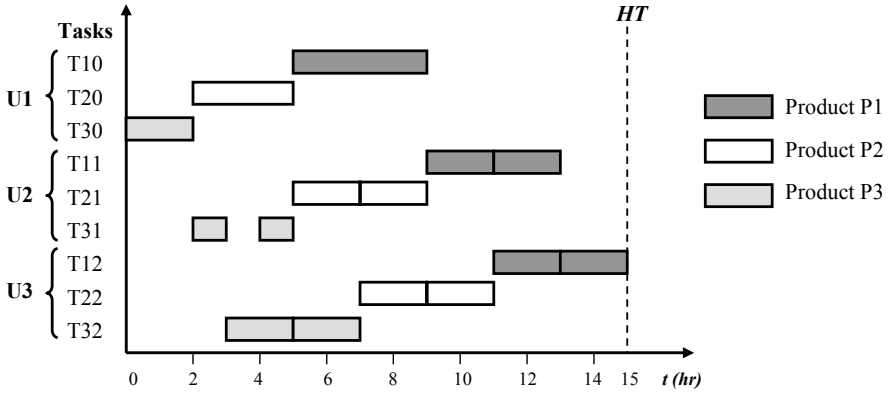


Figure 13: Task Gantt chart for Example 1.

5.2. Example 2

The batch plant of Figure 14 is used for the production of four final products (P1, P2, P3 and P4) from 6 raw materials (F1, F2, … F6). It involves 8 units, 27 states and 19 tasks. Some of these tasks can be performed in more than one unit. Unlimited storage is available for raw materials and final products, finite intermediate storage is available for states S20, S30, S31, S61 and S71, no intermediate storage is available for states S40, S50 and S60 and zero-wait policy applies for states S10, S21, S22 and S70. Sufficient amounts of raw materials and zero amounts of all other states are available at the beginning of the scheduling horizon. The objective is to minimize the makespan for a fixed production of 5 tons for all products. Unit and storage tank capacity data as well as processing time data are given in Appendix E.

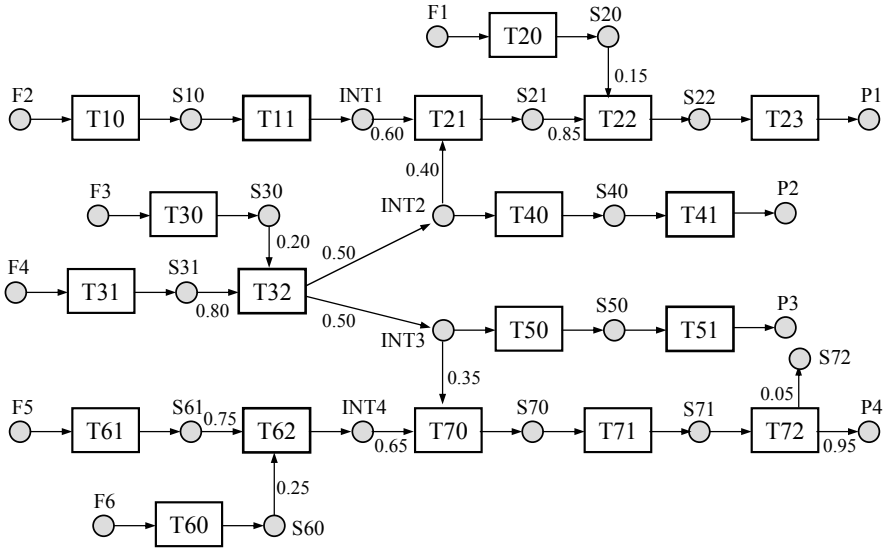


Figure 14: Process network for Example 2.

Since there are no similar equipment and units, no integer cuts of type III are used. Since NIS policy applies for state S60, and tasks T60 and T61 can be carried in only one unit, we add an integer cut of type II that enforces the condition that in any solution the number of copies of task T60 is equal to the number of copies of task T61,

$$\sum_c Z_{T60,c} = \sum_c Z_{T61,c}$$

The same condition is imposed for the zero wait states, i.e. for the following pairs of tasks: (T10, T11), (T21, T22), (T22, T23) and (T71, T72). Moreover, since the objective is the minimization of makespan the integer cuts I exclude not only the current assignment, but also every superset of the current assignment [equation (41)].

Assuming that we can have at most 4 copies of each task, the optimal solution of 15 hours is found in 5 iterations. The assignment that gives the optimal solution is found in the first iteration with a lower bound of 14 hours. Successive feasibility CP problems are solved for this assignment, and a feasible schedule with a makespan of 15 hours is found in the second subproblem. The subsequent master problems give solutions with a lower bound on the makespan equal to 14 hours, but none of these assignments yields a feasible schedule with makespan shorter than 15 hours. The fifth MILP is infeasible, which means that there are no more assignments that can meet the given demand. The total computational time is 1.80 CPU seconds, from which 0.03 seconds are spent in preprocessing, 0.70 seconds are spent for the master problem (approximately 0.14 sec for each MILP), and 1.07 seconds are spent for all the CP subproblems. The Gantt chart of the optimal solution is shown in Figure 15, where the batch size of each task is given in parentheses.

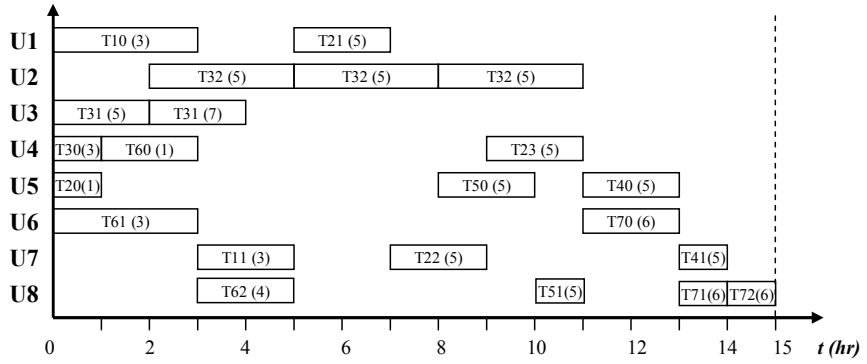


Figure 15: Equipment Gantt chart for Example 2 without changeovers.

We next solve Example 2 assuming a changeover time of 1 hour between different tasks performed in the same unit. An optimal solution of 16 hours was obtained in 5 iterations and 2.66 of CPU seconds (0.03 for preprocessing, 0.71 for MILPs and 1.92 CPU seconds for all CP problems). Note that the increase in computational effort for changeover times is very small. The equipment Gantt chart of the solution is shown in Figure 16. The schedule of Figure 16 is different from the schedule of Figure 15 in that (a) task T60 starts in unit U4 at *t=3* instead of *t=2* due to the changeover time, (b) task T62 starts in unit U8 at *t=4* instead of *t=3* because the intermediate S60 produced by T60 becomes available 1 hour later, and (c) task T72 starts at *t=15* (instead of *t=14*) due to the changeover time needed in unit U8.

Note that when changeover times are included, the CP formulation of the subproblem remains practically the same. The only differences are that we should now define a transition type (TT) for the activities performed in the unit for which changeover times are required, through equation (45), provide the matrix of changeover times and define UnaryResources and Activities through equations (46) and (47), respectively, instead of equations (11) and (14):

Tasks TT[t in Tasks,c in Copies] = t; (45)

UnaryResource Equipment[Units] (m); (46)

Activity Task[t in Tasks, c in Copies] (DurA[t,c]) transitionType TT[t,c]; (47)

where TT is the type of transition and m is the matrix of changeover times (i.e. it is defined for every pair of tasks).

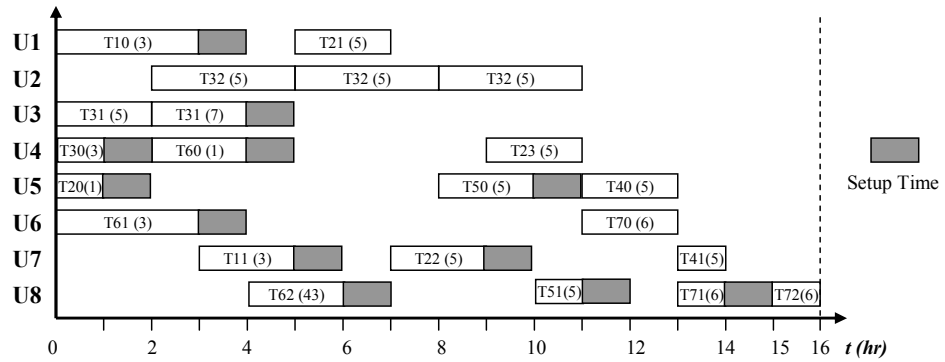


Figure 16: Equipment Gantt chart for Example 2 with changeovers.

6. Computational Issues

6.1. Computational Results and Comparisons

In this section we present computational comparisons with the STN model of Maravelias and Grossmann (2003a) which appears to be among the most effective continuous-time MILP models for general scheduling problems. This model is solved using GAMS 20.7/CPLEX 7.5, while the hybrid scheme was modeled in OPL Studio 3.5, using CPLEX 7.5 for the master MILP and ILOG's Solver 5.2 for the CP subproblem. The default CPLEX and Solver options have been used in all cases. A Pentium III PC at 670 MHz running Redhat Linux is used for both approaches. The comparison is made for both constant and variable processing times, for six different instances of Example 1. In instances Pr1, Pr2 and Pr3 we maximize the total production over a fixed time horizon of 15, 20 and 25 hours, respectively. In instances Ms1, Ms2 and Ms3 we minimize the makespan for fixed demands. The demand for products P1/P2/P3 in instances Ms1, Ms2 and Ms3 are 4/5/6, 5/6/8 and 5/8/10 tons, respectively. Suffices c and f are used to characterize the instances with constant and variable, respectively, processing times. For the hybrid scheme, we have used a maximum number of five copies in all instances (i.e. $|C_i|=5$).

Note that in continuous-time MILP models, the quality of the solution obtained and the computational effort required depend heavily on the number of time intervals in which the time horizon is divided. The minimum number of time points needed to obtain the optimal solution is usually found through an iterative procedure where the number of time points is increased until there is no improvement in the objective function. This procedure, however, does not guarantee that the globally optimal solution is found, as one might get the same solution when the number of time points is increased by one, but then obtain a better solution when the number of time points is increased by two. This feature is not required in the MILP master problem of the proposed hybrid method.

The computational results of the two methods for constant processing times are reported in Table 5. For the maximization of production, the MILP model of Maravelias and Grossmann obtained the optimal solution in all instances with 10, 12 and 15 time points, respectively. Reasonable computational effort was required for

instances Pr1c and Pr3c (58.5 and 523.6 CPU secs, respectively), while more than 4.5 CPU hours were required for Pr2c. It seems that the reason for the large computational requirements of instance Pr2c is the existence of many "equivalent" solutions. Specifically, the earliest start time for unit U3 is *t=3* hours and the processing times of all tasks that can be performed in unit U3 is 2 hours. This implies that for a total time horizon of 20 hours (as in Pr2c) the total available time for processing time in unit U3 is 17 hours, while the maximum number of tasks assigned to U3 is 8. Since the total processing time in unit U3 is 16 hours (8 tasks * 2 hours) and the total available time is 17 (=20-3) hours, there are several different solutions with exactly the same objective. These solutions can be obtained by moving the tasks in U3 (forward or backward) by one hour. While these solutions are different solutions for the MILP models, they correspond to the same solution of the master problem of the proposed approach and thus examined only once.

Table 5: Computational results for Example 1 for constant processing times.

			MILP STN Model			MILP/CP Hybrid Scheme		
Case	H (hours)	Obj ($10^3)	Time Points	Nodes	CPU sec	Obj ($10^3)	# iter's	CPU sec
Pr1c	15	12	10	739	58.5	12	4	0.48
Pr2c	20	16	12	150,922	16,749.2	16	3	0.32
Pr3c	25	22	15	1,570	523.6	22	4	0.43
	Demand (tons)	Obj (hr)				Obj (hr)		
Ms1c	4/5/6	23*	13	171,251	36,000	19	1	0.22
Ms2c	5/6/8	- **	16	111,700	36,000	23	1	11.20
Ms3c	5/8/10	- **	19	57,000	36,000	27	1	0.37

* Suboptimal solution: Gap 35%
** No integer solution found

For the minimization of makespan for fixed demands, the MILP model is intractable. Instance Ms1c is the only one for which an integer solution is found with 13 time points, while instances Ms2c and Ms3c were solved for various time grids (i.e. no of time points) but no integer feasible solution was found in 36,000 CPU seconds. The computational statistics reported in Table 5, correspond to the MILP with the minimum number of time points that can represent the optimal solution found by the hybrid algorithm. The proposed algorithm, on the other hand, obtained the optimal solution for all instances of both objectives. For the maximization of profit instances Pr1c, Pr2c and Pr3c it required 0.48, 0.32 and 0.43 CPU seconds, respectively. It is, therefore, more than two orders of magnitude faster that the MILP model for instances Pr1c and Pr3c, and more than four orders of magnitude faster for instance Pr2c. Instances Ms1c, Ms2c and Ms3c, for the minimization of makespan with fixed demand, were also solved to optimality in less than 12 seconds. Compared to the results of the MILP model, we see that problems that were unsolvable with continuous-time MILP models are solved in seconds with the proposed scheme.

The problem becomes more difficult for both methods when variable processing times are used. Due to the large number of time points and the excessive computational effort needed to solve large problems, the MILP model cannot find the optimal solution in none of the Pr1v, Pr2v and Pr3v instances. Feasible solutions are found for certain time grids, but when finer time grids are used the model becomes intractable. To illustrate we report in Table 6 the computational statistics of the MILP model of Maravelias and Grossmann for various time grids, for instance Pr3v. As shown, models with up to 11 time points are solved with reasonable computational effort, but when 12 or more time points are used, the model becomes intractable. Specifically, a solution with an objective value of 14.0 is obtained when 10 time points are used; an improved solution with an objective value of 16.0 is obtained with 11 time points; but no further improvement is possible, because the problems with more than 11 time points can only yield suboptimal solutions within the time limit of 36,000 CPU seconds. Hence, the best solution for the MILP is 16 (obtained with 11 time points). This is a feature common to all continuous-time MILP models, and restricts their use to mid-size problems.

Table 6: Computational statistics of the MILP model of Maravelias and Grossmann (2003a) for Pr3v.

Time Points	10	11	12	13	14
Objective	14.0	16.0	16.0 *	13.0 *	12.5 *
Best Bound	-	-	17.98	19.98	21.95
Gap (%)	-	-	12.4	53.7	75.6
Nodes	37	1,669	173,893	110,531	86,198
CPU sec	16.9	356.0	36,000	36,000	36,000

* Optimality not proved

The computational comparison of the two methods for variable processing times is given in Table 7. For the maximization of production, we were not able to find the optimal solution using the continuous-time MILP formulation because when more than 12 time points were used the resulting models could not be solved to optimality. Thus, for each instance we report the statistics of the MILP that obtained the best solution in 36,000 CPU seconds (which is significantly smaller than the sum of the CPU times spent for all the MILP models with different number of points). The problem of minimizing makespan for fixed demand proved to be even more difficult for the MILP model as we were not able to get a single integer solution for any of the three instances. In Table 7 we report the computational statistics of the MILP with the smallest number of time points that was not infeasible for the specified demand.

Table 7: Computational results for Example 1 for variable processing times.

			MILP STN Model ♦			MILP/CP Hybrid Scheme ♦♦		
Case	H (hours)	Obj (10^3)	Time points	Nodes	CPU sec	Obj (10^3)	# iter's	CPU sec
Pr1v	15	11.9*	12	215,214	36,000	12.0	31	8.85
Pr2v	20	12.0 **	12	175,799	36,000	16.5	125	297.76
Pr3v	25	16.0	11	1,669	356.0	20.5	60	36,000
	Demand (tons)	Obj (hr)				Obj (hr)		
Ms1v	4/5/6	- ***	13	98,400	36,000	19.7	31	11.28
Ms2v	5/6/8	- ***	14	58,500	36,000	23.8	33	27.67
Ms3v	5/8/10	- ***	15	52,400	36,000	28.1	38	3,120.79

♦ Statistics of the MILP that gave the best solution in 36,000 CPU-s.
♦♦ Best solutions with 5 copies for each task.
* Suboptimal solution: Gap 0.05%
** Suboptimal solution: Gap 32.9%
*** No integer solution found

In the proposed hybrid approach we need to scale and approximate the parameters and scale the time horizon when we have variable processing times (see section 4.6.4.). In order to reduce the domains of the variables we have overestimated some processing times. The parameters that we used for this approximation are given in Table E4 of Appendix E. Nevertheless, the hybrid scheme yields better solutions than the MILP model in all instances. For the maximization of profit we obtained better solutions than the ones obtained by the MILP model, and with significantly less computational effort for instances Pr1v and Pr2v. As explained above, the MILP for instance Pr3v yields a suboptimal solution (profit = 16) in 356 CPU seconds but it becomes intractable for finer time partitions (Table 6) and no better solution can be found. The proposed scheme, in contrast, yields a much better solution (profit = 20.5). For the minimization of makespan we obtained the optimal solution (in terms of tasks) but with slightly overestimated makespan due to the scaling and the overestimation of processing times. As shown, and especially for the minimization of makespan, the proposed algorithm enables us to solve problems that were unsolvable with the existing tools.

Finally, in Table 8 we present the number of iterations and the CPU time required for the preprocessing step, the solution of all MILP master problems and the solution of all CP subproblems. As shown, computational requirements for preprocessing are negligible for all instances. Note, however, that for these examples we only calculate EST and LFT of tasks and units (i.e. Preprocessing I of Figure 3) but we do not derive any integer cuts of type II and III (i.e. Preprocessing II of Figure 3) that could be potentially more time

consuming. The ratio of the total time spent for the solution of the MILP master-problem over the total time spent for the solution of the CP subproblem is variable: it is close to 1 for the easiest problems (instances Pr1c, Pr2c, Pr3c and Ms1c) but it decreases as problems become more difficult; it is almost three orders of magnitude smaller for instance Pr3v. An interesting observation is that the time needed for the solution of a single MILP problem does not increase significantly as the problems becomes more difficult: it varies between 0.055 CPU-s per iteration (Pr3c) up to 0.5 CPU-s per iteration (Pr3v). Hence, the factor that seems to affect the computational efficiency of the proposed scheme the most is the solution of the CP subproblem. Note that in CP the user can customize the search by choosing the order of variables at which branching is performed, constraint propagation rules (disjunctive, edge-finding, etc.) and search strategy (depth first, best first, slice-based, depth-bounded discrepancy, etc.). Hence, we were able to greatly enhance the solution of the CP subproblem by fine tuning, but for comparison reasons here we report the CPU requirements needed when the standard CP and MILP options are used both for the MILP of Maravelias and Grossmann (2003a) and the proposed hybrid scheme.

Table 8: Computational results for Example 1 for variable processing times.

Problem	Instance	Iterations	Detailed Computational Requirements (CPU-s)			
			Pre-processing	MILP	CP	Total
Example 1						
Constant	Pr1	4	0.02	0.26	0.20	0.48
Processing	Pr2	3	0.02	0.18	0.12	0.32
Times	Pr3	4	0.03	0.22	0.18	0.43
	Ms1	1	0.01	0.1	0.11	0.22
	Ms2	1	0.02	0.1	11.08	11.20
	Ms3	1	0.02	0.09	0.26	0.37
Variable	Pr1	31	0.02	6.74	2.09	8.85
Processing	Pr2	125	0.02	54.02	243.72	297.76
Times	Pr3	60	0.02	29.98	35,970.00	36,000
	Ms1	31	0.02	4.05	7.21	11.28
	Ms2	33	0.02	4.66	22.99	27.67
	Ms3	38	0.02	5.33	3,115.44	3,120.79
Example 2						
	w/o setups	5	0.03	0.70	1.07	1.80
	w/ setups	5	0.03	0.71	1.92	2.66

6.2. Remarks

6.2.1. Feasibility Emphasis in MILP Solvers

In order to obtain a single feasible solution for the minimization of makespan using the MILP model of Maravelias and Grossmann, we could have used the CPLEX option for emphasis on feasibility. When we used this option we were indeed able to get feasible solutions for more instances, but these solutions were very poor and it was not possible to improve them in 36,000 CPU seconds.

6.2.3. Discrete-time STN Formulations

The reason we chose to compare the proposed method with a continuous-time STN model is because continuous-time STN models are, in principle, more general than discrete-time STN models. Despite the approximations needed, however, discrete-time models are widely used in practice because they are very effective. Variable processing times can be handled through the introduction of additional binary variables (Kondili et al., 1993). Moreover, an iterative scheme was recently proposed (Maravelias and Grossmann, 2003c) to address the problem of minimization of makespan of multipurpose batch plants using a discrete-time representation.

6.2.3. Application of Preprocessing in MILP Formulations

It is legitimate to argue that for a fair comparison between the proposed method and standalone MILP models, preprocessing must be applied to both approaches. However, it is not clear how to apply the preprocessing described in this paper for MILP models. Specifically, it is not clear how to relate EST and LST of tasks and units with the W_{in} binaries used in continuous-time MILP models to denote the start of task i at time point n. Consider, for example, that the EST of unit j is $t=3$ hours. Since, in continuous-time models we do not know a priori when a time point occurs, we cannot fix any of the W_{in} binaries, other than the one that corresponds to $t=0$. Given that point $n=0$ corresponds to the start of the scheduling horizon (i.e. $t=0$), if the first time point ($n=1$) corresponds to $t=2$ (i.e. $T_1=2$) we could have fixed $W_{i1} = 0 \ \forall i \in I(j)$, but if the first point corresponds to $t=4$ this constraint is incorrect. In discrete-time STN models (where we know when a time point occurs) it is possible to fix binaries to zero, but these constraints are usually automatically satisfied by the MIP formulation. Moreover, most commercial solvers that are used for the solution of the MILP models have very efficient preprocessing routines.

6.2.4. Application of Integer Cuts in MILP Formulations

Similarly, it is not straightforward how to lift an integer cut in the space of W_{in} binaries. Consider, for example, the following cut:

$$Z_{A1} + Z_{B1} - Z_{C1} \leq 1 \tag{48}$$

which excludes a solution in which tasks A and B are performed and task C is not performed.
Can we write this cut using binaries W_{An}, W_{Bn} and W_{Cn}? In general, we cannot replace Z_i binaries by the summation of W_{in} binaries over n, because such a cut excludes more assignments:

$$\Sigma_n W_{An} + \Sigma_n W_{An} - \Sigma_n W_{An} \leq 1 \tag{49}$$

The integer cut in (49), for example, excludes the possibly feasible assignment where task A is performed twice and tasks B and C are not performed at all. The only case where cuts can be lifted in the space of the W_{in} binaries is when tasks can be performed only once (in this case equation (49) is correct). This case, however, is a very restricted subclass of multipurpose batch plants; actually, one of the main advantages of STN representation is that it is not restricted to this subclass of problems. Moreover, for this subclass of problems there are several things that can be done to enhance the performance of both the proposed method and MILP models.
Finally, another issue that has to be resolved is how we acquire these or other types of cuts from a standalone MILP representation. In the proposed framework we identify assignments in the master problem and check feasibility in the subproblem. What type of iterative scheme can be used with continuous-time MILP models? Run a MILP solver for some time and check what binary combinations are infeasible? This would be practically impossible because there is no direct way to extract this information from a MILP solver, the space of binary variables in MILP models is much bigger than the one in the master problem of the proposed approach, and has never been proposed in the past. An alternative would be to use the proposed framework to derive some integer cuts, lift them in the space of W_{in} binaries (only for the special subclass discussed above), and add these cuts in the MILP formulation. In this case, however, we would still need the proposed hybrid scheme. Overall, although pre-processing and integer cuts could potentially benefit standalone MILP formulations as well, currently there is no direct way to apply these enhancements in MILP STN-based formulations.

7. Conclusions

A hybrid MILP/CP framework for STN scheduling problems has been presented in this paper. The proposed framework integrates traditional MILP and CP techniques, exploiting their complementary strengths. Mixed-Integer Programming is used to identify potentially good assignments of equipment units to tasks and Constraint Programming is used to check feasibility and/or to derive optimal feasible schedules for specific assignments. Various classes of integer cuts are presented to exclude infeasible or previously examined assignments, and a graph-theoretic procedure is developed for pre-processing and domain reduction. The proposed framework is very general as it handles: (a) variable batch sizes and processing times, (b) complex plant configurations with batch splitting, mixing and recycle streams, (c) resource constraints other than equipment, and (d) different storage policies. Moreover, various objective functions can be accommodated. It can also be used as a two-level planning and scheduling tool, where the medium-term planning decisions (type and no of tasks needed and assignment of tasks to units) are made by the master MILP problem and the short-term detailed scheduling decisions are made by the CP subproblem. The proposed method can be extended to almost all batch plant configurations, where the MILP master problem can be *enriched* with constraints that represent the special structure of the problem, and the CP subproblem can be simplified if some of the features are not needed. Finally, the computational results showed that for some classes of problems the proposed MILP/CP hybrid algorithm was orders of magnitude faster than MILP models, enabling us to solve problems that are practically unsolvable with existing tools.

Acknowledgements

The authors gratefully acknowledge financial support from the National Science Foundation under Grant ACI-0121497.

Nomenclature

Indices	
i	Tasks
j	Equipment units
c	Copies of task
r	Resource categories (utilities)
s	States
g	Classes of *similar* tasks
d	Orders
Sets	
ZW	Set of tasks producing materials for which zero-wait storage policy applies
FP	Set of final products
INT	Set of intermediate states
$I(j)$	Set of tasks that can be scheduled on equipment j
$I(s)$	Set of tasks that use state s as input
$I(FP)$	Set of tasks that produce final products
$I(g)$	Set of tasks that belong to class g
$O(s)$	Set of tasks that produce state s
$SI(i)$	Set of states consumed by task i
$SO(i)$	Set of states produced by task i
Parameters	
H	Time horizon
α_i	Fixed duration of task i

β_i Variable duration of task i
γ_{ir} Fixed amount or utility r required for task i
δ_{ir} Variable amount of utility r required for task i
ρ^I_{is} / ρ^O_{is} Mass balance coefficient for the consumption/production of state s in task i
$S0_s$ Initial amount of state s
C_s Storage capacity for state s
R^{MAX}_r Upper bound for utility r
B^{MIN}_i / B^{MAX}_i Lower/upper bounds on the batch size of task i
ζ_s Price of state s
AD_d Amount due for order d
TD_d Due date of order d
EST_i / ST_i / LFT_i Earliest Start Time / Shortest Tail / Latest Finish Time of task i
EST_j / ST_j / LFT_j Earliest Start Time / Shortest Tail / Latest Finish Time of unit j

Variables of MILP Master Problem (MP)

Binary Variables

Z_{ic} $=1$ if batch c of task i is performed
Y_{gcn} $=1$ if there are n c^{th} copies of tasks in $I(g)$

Continuous Variables

MS Makespan
D_{ic} Duration of copy c of task i
B_{ic} Batch size of copy c of task i
$\hat{S}_s$ Amount of state s

Variables and constructs of CP Subproblem (SP)

Variables

D_{ic} Duration of copy c of task i
B_{ic} Batch size of copy c of task i
B^I_{ics} / B^O_{ics} Amount of state s consumed/produced by copy c of task i
R_{icr} Amount of utility r consumed by copy c of task i

Constraint Programming Constructs

MS Dummy activity that appears at the end of the scheduling horizon
Task[i,c] Activity that represents the c^{th} copy of task i with duration D_{ic}
Order[d] Activity used to represent orders with due dates
Unit[j] Unary resource that represents the equipment unit j
Utility[r] Discrete resource that represents utility r with maximum capacity R_r^{MAX}
State[s] Reservoir that represents state s, with initial inventory $S0_s$ and storage capacity C_s.

Variables of MILP model of Maravelias and Grossmann (2003a)

Binary Variables

Ws_{in} $=1$ if task i starts at time point n
Wp_{in} $=1$ if task i is being processed at time point n
Wf_{in} $=1$ if task i finishes at or before time point n

Continuous Variables

T_n Time that corresponds to time point n
Ts_{in} Start time of task i that starts at time point n
Tf_{in} Finish time of task i that starts at time point n
D_{in} Duration of task i that starts at time point n
Bs_{in} Batch size of task i that starts at time point n
Bp_{in} Batch size of task i that is processed at time point n
Bf_{in} Batch size of task i that finishes at or before time point n

B^I_{isn} Amount of state *s* used as input for task *i* at time point *n*
B^O_{isn} Amount of state *s* produced from task *i* at or before time point *n*
S_{sn} Amount of state *s* available at time point *n*
R^I_{irn} Amount of utility *r* consumed at time point *n* by task *i*
R^O_{irn} Amount of utility *r* released at or before time point *n* by task *i*
R_{rn} Amount of utility *r* consumed at time point *n*

Appendix A: ILOG's OPL Studio 3.5 Modeling Language

In this paper we model CP subproblems using the modeling language of ILOG's OPL Studio 3.5, which has a number of global constraints and special constructs that are specifically developed for scheduling applications. The special constructs are:

Activity: It is equivalent to a job or a task. There are three variables associated with each activity *a*: its start time (*a.start*), its duration (*a.duration*) and its end time (*a.end*). The user has to define *a.duration*.
Unary Resource: It is a resource with capacity equal to 1 that can be used by only one activity at a time. It is used to model equipment units.
Discrete Resource: It is a resource with (integer) capacity *R>1* and it is used to model discrete (e.g. manpower) and continuous resources (cooling water). Its constant capacity *R* is user-defined.
Reservoir: It is an entity that can be consumed/produced by a task and it is used to model the chemicals that are consumed/produced by tasks (activities). Its capacity *C* and initial amount *S0* are user-defined.

Let *TASK* be an activity, *UNIT* a unary resource, *UTILITY* a discrete resource and *STATE* a reservoir. The global constraints used in the CP subproblem are:

TASK requires UNIT *(A1)*

TASK requires R_{TASK} *UTILITY* *(A2)*

TASK consumes B^I *STATE* *(A3)*

TASK produces B^O *STATE* *(A4)*

where *requires*, *consumes* and *produces* are OPL's special global constraints.
Constraint (A1) is an assignment constraint that assigns and sequences activity *TASK* to unary resource *UNIT*, taking into account that at most one activity can be assigned to *UNIT* at any given time point. Constraint (A2) allocates R_{TASK} units of discrete resource *UTILITY* to activity *TASK*, taking into account that at most *R* units of *UTILITY* can be allocated to all activities at any time point. Constraints (A3) and (A4) are used for the calculation of the amount of reservoir *STATE* consumed and produced by activity *TASK*, taking into account that the level of reservoir *STATE* at any time cannot be less than zero and cannot exceed its maximum capacity *C*.

Appendix B: MILP Model of Maravelias and Grossmann (2003a)

Among the continuous-time models the model of Maravelias and Grossmann (2003a) appears to be quite general in terms of the plant configurations and computationally among the most effective. This model is

presented here for reference and computational comparisons with the proposed hybrid MILP/CP decomposition scheme.

Binaries Ws_{in} (Wf_{in}) are 1 if task i starts at time point n (finishes at or before time point n). Variable T_n is the time at which time point n occurs, D_{in} is the duration of task i that starts at time point n, and Ts_{in} (Tf_{in}) is the start (finish) time of task i. Variables Bs_{in}, Bp_{in} and Bf_{in} correspond to the batch size of task i that starts at, is being processed at, and finishes at or before time point n, respectively. Variable B^O_{isn} (B^I_{isn}) represents the amount of state s produced (consumed) by task i at time point n. Variable R^I_{irn} (R^O_{irn}) represents the amount of renewable resource (utility) r consumed (released) by task i at time point n. The amount of state s stored at point n is denoted by S_{sn}:

Assignment Constraints

$$\sum_{i\in I(j)}\sum_{n'\le n}(Ws_{in'}-Wf_{in'})\le 1 \quad \forall j,\forall n \tag{B1}$$

$$\sum_{n} Ws_{in}=\sum_{n} Wf_{in} \quad \forall i \tag{B2}$$

$$\sum_{i\in I(j)} Ws_{in}\le 1 \quad \forall j,\forall n \tag{B3}$$

$$\sum_{i\in I(j)} Wf_{in}\le 1 \quad \forall j,\forall n \tag{B4}$$

Calculation of Duration and Finish Time of Tasks

$$D_{in}=\alpha_i Ws_{in}+\beta_i Bs_{in} \quad \forall i,\forall n \tag{B5}$$

$$Tf_{in}\le Ts_{in}+D_{in}+H(1-Ws_{in}) \quad \forall i,\forall n \tag{B6}$$

$$Tf_{in}\ge Ts_{in}+D_{in}-H(1-Ws_{in}) \quad \forall i,\forall n \tag{B7}$$

$$Tf_{in}-Tf_{in-1}\le H\cdot Ws_{in} \quad \forall i,\forall n \tag{B8}$$

$$Tf_{in}-Tf_{in-1}\ge \alpha_i\cdot Ws_{in} \quad \forall i,\forall n \tag{B9}$$

Time-matching Constraints

$$Ts_{in}=T_n \quad \forall i,\forall n \tag{B10}$$

$$Tf_{in-1}\le T_n+H(1-Wf_{in}) \quad \forall i,\forall n \tag{B11}$$

$$Tf_{in-1}\ge T_n-H(1-Wf_{in}) \quad \forall i\in ZW(i),\forall n \tag{B12}$$

Batch-size, Storage Constraints and Material Balances

$$B_i^{MIN}Ws_{in}\le Bs_{in}\le B_i^{MAX}Ws_{in} \quad \forall i,\forall n \tag{B13}$$

$$B_i^{MIN}Wf_{in}\le Bf_{in}\le B_i^{MAX}Wf_{in} \quad \forall i,\forall n \tag{B14}$$

$$B_i^{MIN}\Big(\sum_{n'<n}Ws_{in'}-\sum_{n'\le n}Wf_{in'}\Big)\le Bp_{in}\le B_i^{MAX}\Big(\sum_{n'<n}Ws_{in'}-\sum_{n'\le n}Wf_{in'}\Big) \quad \forall i,\forall n \tag{B15}$$

$$Bs_{in-1}+Bp_{in-1}=Bp_{in}+Bf_{in} \quad \forall i,\forall n \tag{B16}$$

$$B^I_{isn}=\rho_{is}Bs_{in} \quad \forall i,\forall n,\forall s\in SI(i) \tag{B17}$$

$$B^O_{isn}=\rho_{is}Bf_{in} \quad \forall i,\forall n,\forall s\in SO(i) \tag{B18}$$

$$S_{sn}=S_{s,n-1}+\sum_{i\in O(s)}B^O_{isn}-\sum_{i\in I(s)}B^I_{isn} \quad \forall s,\forall n \tag{B19}$$

$$S_{sn}\le C_s \quad \forall s,\forall n \tag{B20}$$

Utility Constraints

$$R^I_{irn} = \gamma_{ir} Ws_{in} + \delta_{irs} Bs_{in} \quad \forall i, \forall r, \forall n \tag{B21}$$

$$R^O_{irn} = \gamma_{ir} Wf_{in} + \delta_{irs} Bf_{in} \quad \forall i, \forall r, \forall n \tag{B22}$$

$$R_{rn} = R_{rn-1} - \sum_i R^O_{irn-1} + \sum_i R^I_{irn} \quad \forall r, \forall n \tag{B23}$$

$$R_{rn} \le R_r^{MAX} \quad \forall r, \forall n \tag{B24}$$

Ordering of Time Points

$$T_1 = 0 \tag{B25}$$

$$T_{|n|} = H \tag{B26}$$

$$T_{n+1} \ge T_n \quad \forall n \tag{B27}$$

Tightening Valid Inequalities

$$\sum_{i \in I(j)} \sum_n D_{in} \le H \quad \forall j \tag{B28}$$

$$\sum_{i \in I(j)} \sum_{n' \ge n} D_{in'} \le H - T_n \quad \forall j, \forall n \tag{B29}$$

$$\sum_{i \in I(j)} \sum_{n' \le n} (Wf_{in'} \alpha_i + Bf_{in'} \beta_i) \le T_n \quad \forall j, \forall n \tag{B30}$$

$$\max Z = \sum_s S_{sn=|N|} \zeta_s \tag{B31}$$

$$Ws_{in},\ Wf_{in} \in \{0,1\},\ Bs_{in},\ Bp_{in},\ Bf_{in},\ SS_{sn},\ S_{sn},\ T_n,\ Ts_{in},\ Tf_{in},\ D_{in},\ B^I_{isn},\ B^O_{isn}\ R^I_{irn},\ R^O_{irn},\ R_{rn} \ge 0 \tag{B32}$$

Constraints (B1) – (B4) are the assignment constraints and constraints (B5) – (B9) are used for the calculation and tightening of the duration, D_{in}, and the finish time, Tf_{in}, of task i. The elimination of start time, Ts_{in}, is accomplished through (B10), while constraints (B11) and (B12) are used for time matching between finish time, Tf_{in}, and T_n. Constraints (B13) – (B18) are used for the bounding of batch sizes (Bs_{in}, Bp_{in} and Bf_{in}) and the amount of state s produced B^O_{isn} (consumed B^I_{isn}) by task i at time point n. Constraint (B19) is the mass balance for state s at time n, and constraint (B20) is a capacity constraint, where C_s is the capacity of the storage tank. The amount R^I_{rn} of renewable resource r required by task i starting at time point n is calculated by constraint (B21). The same amount R^O_{rn} is "released" when task i finishes, and is calculated by constraint (B22). The total amount of resource r required at time n is calculated in (B23) and bounded, not to exceed the maximum availability R_r^{MAX}, by (B24). Equations (B25) – (B27) define the start and the end of the time horizon and enforce an ordering among time points. The addition of valid inequalities (B28), (B29) and (B30) tightens the LP relaxation and significantly reduces the size of the branch-and-bound tree. The MILP model (M) for the maximization of profit over a fixed time horizon H consists of equations (B1) to (B32).

If the objective is to minimize the makespan MS for fixed demand, the following changes need to be made:

(a) The objective function is,

$$\min\ MS \tag{B31'}$$

(b) Addition of constraint (33) that enforces that the demand is met,

$$S_{sn} \ge d_s \quad \forall s, n = |N| \qquad \text{or} \qquad S_{sn} = d_s \quad \forall s, n = |N| \tag{B33}$$

where d_s is the demand for state s at the end of the horizon.

(c) The length of the fixed time horizon, H, is replaced by the makespan, MS, in constraints (B26), (B28) and (B29). The parameter H is used in all other constraints as an upper bound on the makespan MS.

The model (M') for the minimization of makespan consists of equations (B1) – (B30), (B31') and (B32) - (B33). Finally, a generalization of model (M) that accounts for due dates and delivery dates has been proposed by Maravelias and Grossmann (2003b).

Appendix C: Derivation of MILP Master Model (MP)

In order to show that the master problem (MP) of the proposed scheme is a relaxation of the model (M) of Maravelias and Grossmann (2003a), we first need to clarify the relation between copies $c \in C_i$ and time points $n \in N$. In the model (M) the authors divide the time horizon into $|N-1|$ time periods, and each task can start at the beginning of any of these intervals ($Ws_{in}=1$ and $Bs_{in} \neq 0$), i.e. can take place more than once. The number NB_i^M of batches of task i can then be calculated from,

$$NB_i^M = \sum_{n \in N} Ws_{in} \quad \forall i \in I \tag{C1}$$

In the master problem (MP) of the proposed approach we use the notion of "copies" of task i, i.e. we postulate, through equation (1), a maximum number of copies (batches) $|C_i|=C_i^{MAX}$ for each task i. The binary Z_{ic} is equal to 1 if copy c of task i takes place [subject to constraint (8) that is a symmetry-breaking constraint]. The number NB_i^{MP} of batches of task i is defined by,

$$NB_i^{MP} = \sum_{c \in C_i} Z_{ic} \quad \forall i \in I \tag{C2}$$

For a solution of model (M) to be mapped onto a solution of model (MP), it is necessary (but not sufficient) NB_i^{MP} to be equal to NB_i^M. Furthermore, for any solution of (M) we can find an equivalent solution of (MP) where the earliest batch (i.e. the smallest n for which $Ws_{in}=1$) of task i of model (M) corresponds to the first ($c=1$) copy of task i of model (MP), the second earliest batch of (M) corresponds to the second copy of (MP), etc., and, moreover, the duration and batch size of each batch of task i in (M) is equal to the duration and batch size of the corresponding copy of task i in (MP).

Taking into account the relationship between copies $c \in C_i$ of model (MP) and time periods $n \in N$ of model (M), we can say that any solution of model (M) is also feasible for model (MP) because,

1. Constraints (2) are the same as constraints (B28); constraints (B28) have some additional zero terms for the time periods where no task starts.
2. Constraints (3) are a subset of constraints (B5).
3. Constraints (4) are a subset of constraints (B13).
4. By adding constraints (B19) of (M) for all $n \in N$ and state s we get:

$$S_{s0} = S0_s - \sum_{i \in I(s)} B_{is1}^I$$

$$S_{s1} = S_{s0} + \sum_{i \in O(s)} B_{is1}^O - \sum_{i \in I(s)} B_{is1}^I$$

$$S_{s2} = S_{s1} + \sum_{i \in O(s)} B_{is2}^O - \sum_{i \in I(s)} B_{is2}^I$$

…

$$S_{s|N|} = S_{s,|N|-1} + \sum_{i \in O(s)} B_{is|N|}^O$$

or

$$S_{s|N|} = S0_s + \sum_{n=1}^{|N|} \sum_{i \in O(s)} B_{isn}^O - \sum_{n=0}^{|N|-1} \sum_{i \in I(s)} B_{isn}^I$$

Reversing the order of summation and plugging in equations (B17) and (B18) we get:

$$S_{s|N|} = S0_s + \sum_{i\in O(s)} \sum_{n=1}^{|N|} \rho_{is}^O B_{in}^O - \sum_{i\in I(s)} \sum_{n=0}^{|N|-1} \rho_{is}^I B_{in}^I \qquad \text{(C3)}$$

As explained above, for any solution of (M) we can construct a solution of (MP) with the same number of batches, and equal durations and batchsizes. Thus, the internal summations can be replaced by summations over the set of copies of (MP),

$$S_{s|N|} = S0_s + \sum_{i\in O(s)} \sum_{c} \rho_{is}^O B_{in}^O - \sum_{i\in I(s)} \sum_{c} \rho_{is}^I B_{in}^I$$

which is the same as constraint (5) of the master problem, if $\hat{S}_s = S_{s|N|}$, i.e. the variable $\hat{S}_s$ of the master problem (MP) corresponds to the final amount of state s in the (M) model.

5. Constraints (6) are the same as constraints (B33)
6. Constraints (7) are the same as constraints (B20) for n=|N|.
7. Constraints (8) are symmetry breaking constraints that cannot be mapped into the space of (M) model.

Hence, from the above it follows that model (MP) consists either of constraints of model (M), or of constraints that are linear combinations of constraints of model (M). This implies that (MP) is a relaxation of (M) and that any solution of (M) can be "mapped" into a solution of (MP). The master problem (MP), therefore, provides an upper bound to the profit and a lower bound to the cost and the makespan.

Appendix D: Example of Calculation of EST and ST of Tasks

Consider the STN network of Figure D1, where we assume, for simplicity, that the processing times of all tasks are 2 hours ($D_i=2,\ \forall i$) and raw materials RM1, RM2 and intermediate INT3 are available at time t=0.

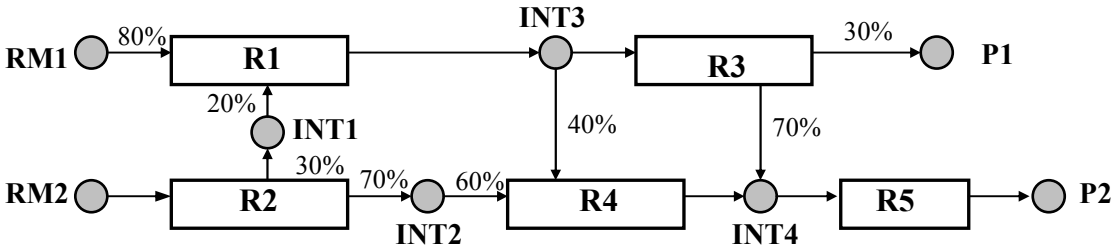


Figure D1: Process network of example for calculation of EST and ST of tasks.

Since RM2 is initially available and R2 consumes only RM2, we have $EST_{R2}=0$. Similarly, the $EST_{R3}=0$. Intermediates INT1 and INT2 are produced only by task R2 and thus they will become available at $EST_{R2}+D_{R2}=2$, i.e. $AT_{INT1}=AT_{INT2}=2$. Task R1 can only start if both RM1 and INT1 are available, which means that $EST_{R1}=max\{AT_{RM1}, AT_{INT1}\}=max\{0,2\}=2$. Similarly, task R4 consumes intermediates INT2 and INT3, and thus $EST_{R3}=max\{AT_{INT3}, AT_{INT2}\}=\{0,2\}=2$. Since intermediate INT3 is available at t=0 task R3 can start immediately ($EST_{R3}=0$). Intermediate INT4 is produced by both R3 and R4 which means that it becomes first available at $AT_{INT4}=min\{EST_{R3}+D_{R3}, EST_{R4}+D_4\}=\{0+2, 2+2\}=2$. Finally, task R5 can start when INT4 is available, i.e. $EST_{R5}=AT_{INT4}=2$. We need not calculate the AT_{P1} and AT_{P2}. The sequence of calculations is shown in Figure D2, where the new calculations are highlighted.

For the calculation of ST of tasks, we have $MT_{P1}=MT_{P2}=0$ and we first calculate $ST_{R5} = 0$, which implies that $MT_{INT4}=D_{R5}=2$. Task R3 is used for the production of both P1 and INT4 and thus we have $ST_{R3} = min\{MT_{P1}, MT_{INT4}\} = 0$. Since task R4 produces only state INT4, $ST_{R4}=MT_{INT4}=2$. State INT3 is used for both tasks R3 and R4, so $MT_{INT3} = min\{ST_{R3}+D_{R3}, ST_{R4}+D_{R4}\} = min\{0+2, 2+2\} = 2$. For state INT2 we have $MT_{INT3} = ST_{R4} + D_4 = 4$. Similarly, $ST_{R1}=2$ and $ST_{R2}=4$. We need not calculate MT_{RM1} and MT_{RM2}.

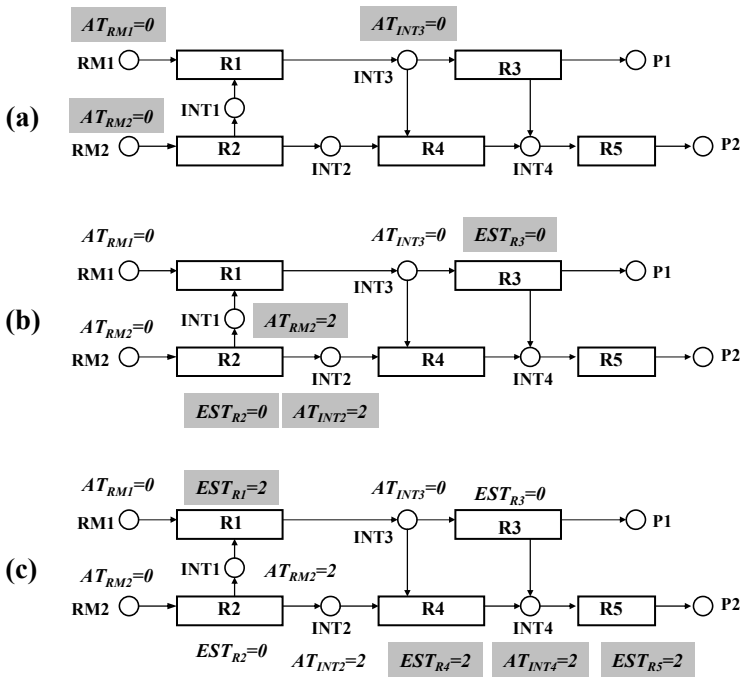


Figure D2: Calculations of for EST of tasks.

Appendix E: Example Data

Table E1: Data for Example 1.

State	F1	F2	F3	S10	S20	S30	S11	S21	S31	P1	P2	P3
Capacity	UIS	UIS	UIS	10	10	10	ZW	ZW	ZW	UIS	UIS	UIS
S_0 (ton)	100	100	100	-	-	-	-	-	-	-	-	-
ζ ($\$10^3$/ton)	-	-	-	-	-	-	-	-	-	1	1	1

Table E2: Data for Example 1 for constant processing times.

Task	T10	T20	T30	T11	T21	T31	T12	T22	T32
Unit	U1	U1	U1	U2	U2	U2	U3	U3	U3
B^{MIN} (ton)	1	1	1	1	1	1	1	1	1
B^{MAX} (ton)	5	5	5	2	2	2	3	3	3
α (hr)	4	3	2	2	2	1	2	2	2

Table E3: Exact data for Example 1 for variable processing times.

Task	T10	T11	T12	T20	T21	T22	T30	T31	T32
Unit	U1	U2	U3	U1	U2	U3	U1	U2	U3
B^{MIN} (ton)	2.5	1	1.5	2.5	1	1.5	2.5	1	1.5
B^{MAX} (ton)	5	2	3	5	2	3	5	2	3
α (hr)	1	0.5	0.5	0.75	0.5	0.5	0.5	0.25	0.5
β (hr/ton)	0.8	1	0.667	0.6	1	0.667	0.4	0.5	0.667

Table E4: Data for Example 1 for variable processing times: parameters used for approximation

Task	T10	T20	T30	T11	T21	T31	T12	T22	T32
Unit	U1	U1	U1	U2	U2	U2	U3	U3	U3
B^{MIN} (500 kg)	5	2	3	5	2	3	5	2	3
B^{MAX} (500 kg)	10	4	6	10	4	6	10	4	6
α (6 min)	10	5	7	8	5	7	5	1	7
β (6 min/ 500 kg)	4	5	3	3	5	3	2	3	3

Table E5: Data for Example 2.

State	F1	F2	F3	F4	F5	F6	S10	S20	S21	S22	S30	S31	S40	S50
Capacity	100	100	100	100	100	100	0	10	0	0	20	20	0	0
S_0 (kg)	100	100	100	100	100	100	-	-	-	-	-	-	-	-

State	S60	S61	S70	S71	S72	IN1	IN2	IN3	IN4	P1	P2	P3	P4
Capacity	0	20	0	10	2.5	100	100	100	100	100	100	100	100
S_0 (kg)	-	-	-	-	-	-	-	-	-	-	-	-	-

Table E6: Data for Example 2.

Task	T10	T11	T20	T21	T22	T23	T30	T31	T32	T40
Duration (hr)	3	2	3	2	2	2	1	2	3	2
Unit	U1	U7	U5	U1	U7	U4	U4	U3	U2	U5
B^{MIN} (ton)	1	1	1	1	1	1	1	1	1	1
B^{MAX} (ton)	6	7	8	6	7	7	7	7	5	8

Task	T41	T50	T51	T60	T61	T62	T70	T71	T72
Duration (hr)	1	2	1	2	3	2	2	1	1
Unit	U2/U7	U5	U2/U8	U4	U6	U8	U6	U8	U8
B^{MIN} (ton)	1	1	1	1	1	1	1	1	1
B^{MAX} (ton)	5/7	8	5/8	7	6	8	6	8	8

References

Baptiste, P.; Le Pape, C.; Nuijten, W. Constrained-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Kluwer Academic Publishers, 2001.

Castro, P.; Barbosa-Povoa, A. P. F. D.; Matos, H. An Improved RTN Continuous-Time Formulation for the Short-term Scheduling of Multipurpose Batch Plants, *Ind. Eng. Chem. Res.*, **2001**, 40, 2059-2068.

Harjunkoski, I.; Grossmann, I.E. Decomposition Techniques for Multistage Scheduling Problems Using Mixed-Integer and Constrained Programming Methods. *Comp. Chem. Engng.*, **2002**, 26, 1533-1552.

Hentenryck, P.V. Constraint Satisfaction in Logic Programming. MIT Press, Cambridge, MA, 1989.

Hentenryck, P.V. Constraint and Integer Programming in OPL. *INFORMS Journal on Computing*, **2002**, 14(4), 345-372.

Hooker, J. Logic Based Methods for Optimization: Combining Optimization and Constraint Satisfaction. John Willey and Sons, Inc. New York, 2000.

Hooker, J.N. Logic, Optimization, and Constraint Programming. *INFORMS Journal on Computing*, **2002**, 14(4), 295-321.

Ierapetritou, M. G.; Floudas, C. A. Effective Continuous-Time Formulation for Short-Term Scheduling. 1. Multipurpose Batch Processes. *Ind. Eng. Chem. Res.*, **1998**, 37, 4341-4359.

Ierapetritou, M. G.; Floudas, C. A. Effective Continuous-Time Formulation for Short-Term Scheduling. 2. Continuous and Semicontinuous Processes. *Ind. Eng. Chem. Res.*, **1998**, 37, 4360-4374.

ILOG OPL Studio 3.5: The Optimization Language, ILOG Inc., **2001**.

ILOG OPL Studio 3.5: The User's Manual, ILOG Inc., **2001**.

Jain, V.; Grossmann, I.E. Algorithms for Hybrid MILP/CP Model for a Class of Optimization Problems. *INFORMS Journal in Computing*, **2001**, 13, 258-276.

Kondili, E.; Pantelides, C. C.; Sargent, R. A General Algorithm for Short-Term Scheduling of Batch Operations – I. MILP Formulation. *Comput. Chem. Eng.* **1993**, 17, 211-227.

Kyu-Hwang Lee; Heung Il Park; In Beum Lee. A Novel Nonuniform Discrete Time Formulation for Short-Term Scheduling of Batch and Continuous Processes. *Ind. Eng. Chem. Res.*, **2001**, 40, 4902-4911.

Maravelias, C.T.; Grossmann, I.E. A New General Continuous-Time State Task Network Formulation for the Short-Term Scheduling of Multipurpose Batch Plants. *Ind. Eng. Chem. Res.*, **2003**, 42(13), 3056-3074.

Maravelias, C. T.; Grossmann, I. E. A Continuous-Time State Task Network Formulation for the Short-Term Scheduling of Multipurpose Batch Plants with Due Dates. PSE 2003, Kunming, China.

Maravelias, C.T.; Grossmann, I.E. Minimization of the Makespan with a Discrete-Time State-Task Network Formulation. *Ind. Eng. Chem. Res.*, **2003**, 42(24), 6252-6257.

Marriott, K. Stuckey, P.J. Introduction to Constraint Logic Programming. MIT Press, Cambridge, MA, 1999.

Mendez, C.A.; Cerda, J. Optimal Scheduling of Resource-Constrained Multiproduct Batch Plant Supplying Intermediates to Nearby End-product Facilities. *Comp. Chem. Engng.*, **2000**, 24, 369-376.

Mendez, C.A.; Henning, G.P.; Cerda, J. Optimal Scheduling of Batch Plants Satisfying Multiple Product Orders with Different Due Dates. *Comp. Chem. Engng.*, **2000**, 24, 2223-2245.

Mendez, C.A.; Henning, G.P.; Cerda, J. An MILP Continuous-Time Approach to Short-Term Scheduling of Resource-Constrained Multistage Flowshop Batch Facilities. *Comp. Chem. Engng.*, **2001**, 25, 701-711.

Mockus, L.; Reklaitis, G.V. Continuous Time Representation Approach to Batch and Continuous Process Scheduling. 1. MINLP Formulation. *Ind. Eng. Chem. Res.* **1999**, 38, 197-203.

Nemhauser, G.L.; Wolsey, L.A. Integer and Combinatorial Optimization, John Willey and Sons, Inc., New York, 1989.

Pantelides, C. C. Unified Frameworks for the Optimal Process Planning and Scheduling. *In Proceedings on the Second Conference on Foundations of Computer Aided Operations* (editors D.W.T. Rippin and J. Hale)*,* **1994**, 253-274.

Papageorgiou, L.G.; Pantelides, C.C. Optimal Campaign Planning/Scheduling of Multipurpose Batch/Semicontinuous Plants. 2. Mathematical Decomposition Approach. *Ind. Eng. Chem. Res.*, **1996**, 35, 510-529.

Pinto, J. M.; Grossmann, I.E. A Continuous Time Mixed Integer Linear Programming Model for Short Term Scheduling of Multistage Batch Plants. *Ind. Eng. Chem. Res.*, **1995**, 34, 3037-3051.

Rodrigues, M. M.; Latre, L.G.; Rodrigues, L.A. Short-term Planning and Scheduling in Multipurpose Batch Chemical Plants: A Multi-level Approach. *Comput. Chem. Eng.* , **2000**, 24, 2247-2258.

Schilling, G.; Pantelides, C. C. A Simple Continuous-Time Process Scheduling Formulation and a Novel Solution Algorithm. *Comput. Chem. Eng.* , **1996**, 20, S1221-1226.

Shah, N.; E.; Pantelides, C. C.; Sargent, R. A General Algorithm for Short-Term Scheduling of Batch Operations – II. Computational Issues. *Comput. Chem. Eng.* **1993**, 17, 229-244.

Zhang, X.; Sargent, R. W. H. The Optimal Operation of Mixed Production Facilities – General Formulation and Some Approaches for the Solution. *Comput. Chem. Eng.*, **1996**, 20, 897-904.