Decomposition Techniques for the Solution of Large-Scale Scheduling Problems

Matthew H. Bassett, Joseph F. Pekny, and Gintaras V. Reklaitis
School of Chemical Engineering, Purdue University, West Lafayette, IN 47907

With increased product specialization within the chemical-processing industries, the ability to obtain production schedules for complex facilities is at a premium. This article discusses ways of quickly obtaining solutions for industrially relevant, large-scale scheduling problems. A number of time-based decomposition approaches are presented along with their associated strengths and weaknesses. It is shown that the most promising of the approaches utilizes a reverse rolling window in conjunction with a disaggregation heuristic. In this method, only a small subsection of the horizon is dealt with at a time, thus reducing the combinatorial complexity of the problem. Resource- and task-unit-based decompositions are also discussed as possible approaches to reduce the problem to manageable proportions. A number of examples are presented throughout to clarify the discussion.

Introduction

Within the chemical-processing industry (CPI) it is becoming increasingly important to economically and efficiently schedule batch facilities. This is due, in large part, to the specialization of products to meet customer demands for numerous small-volume, high value-added products. The ability to quickly and effectively schedule an entire multipurpose facility based on numerous recipes is key to maximizing profitability. For this reason, a great deal of effort has been applied to the problem of production scheduling both in industry and academia. Reviews by Ku et al. (1987) and Musier and Evans (1990) showed many of the recent developments in both areas. Miller et al. (1993) presented a system in use within Dupont for scheduling a number of processes, including polymer blending/extruding. Batch scheduling is also of interest to the pharmaceutical and agrichemical industries where batch processing is the primary production route.

Before going any further, we should formalize what is meant by a production-scheduling problem. In general terms, a production schedule attempts to allocate limited resources in the manner that maximizes a given objective. These resources may be equipment, manpower, raw materials, storage, time, and anything else that is relevant to the problem at hand and available in limited supply. Some common objectives are to maximize profit or minimize production costs.

One method for solving scheduling problems is to represent them as mathematical programs. These programs can then be solved by branch and bound or other solution techniques depending on the formulation type. The use of a mathematical programming paradigm maintains an accurate scorecard (material balances, equipment utilization, etc.) from which future decisions can be based. At the same time, it provides a consistent metric (objective function) by which to measure the effect these decisions have on the schedule. Mathematical programming methods have the potential of being more flexible than specialized heuristics if reliable solution techniques can be developed and can be more cost-effective due to their reusability.

The mathematical program for a scheduling problem can contain as much detail about the process(es) as needed. However, the more detailed the process description and/or the longer the scheduling horizon, the larger the formulation generated. This can lead to formulations that are intractable given available computational resources. A more detailed look at the reasons for this intractability yields insight into solution methods.

As depicted in Figure 1, there are three main complicating components to any scheduling problem: the number of tasks/resources, the number of units, and the length of time.

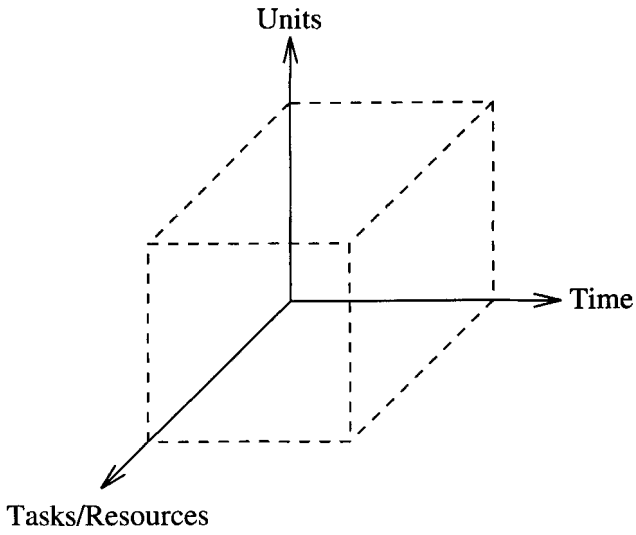

Figure 1. Complexity cube.

One can think of the overall complexity of the problem at hand as being the volume of a polyhedron bounded by the "level" of each component. However, care should be taken not to assume that complexity changes as a simple cubic relationship of these components. So, while, an increase in any one of the components leads to more complex problems, this change in complexity is not necessarily uniform. For example, a problem with multiple products and/or production pathways has a large task/resources component. Allowing equipment sharing or having multiple production sites increases the unit-based component of a problem. Coupling of these two effects over a long horizon leads to a problem of great complexity. Reasonable solution times for general scheduling problems will only be achievable if methodologies are devised to mitigate the levels of one or all of these components.

In this article we present a number of decomposition techniques that have been investigated for the solution of large-scale scheduling problems in batch chemical processes. These techniques reduce the complexity involved by decomposing the problem in time, units, tasks and/or resources. In the following section, two formulations are presented that are used throughout the rest of the article. A number of time-based decompositions are presented, as well as some other possible decomposition techniques. Finally, some conclusions are drawn as to when certain decompositions are indicated or contraindicated.

Formulations

A full description of a scheduling problem consists of product recipes, available equipment, and forecasted demands. This information can be used within a number of scheduling formulations, two of which are presented and utilized throughout this article.

Uniform discretization model formulation

Uniform discretization models (UDMs) for scheduling problems were first proposed as early as 1959 (see Bowman, 1959; Manne, 1960). In terms of industrial-scale problems, very little was done with UDMs at the time, but with the increased capabilities of modern computers, interest in these models has reemerged. Within the chemical-engineering community, UDMs and possible solution methods were reintroduced by Kondili et al. (1988). Since then, significant work utilizing UDMs has been published in the chemical-engineering literature due to the general applicability of the models (Sahinidis and Grossmann, 1991; Kondili et al., 1993; Pekny and Zentner, 1993; Shah et al., 1993; Pantelides, 1994; Zentner et al., 1994).

The UDM formulation solves scheduling problems by discretizing the problem into uniform time intervals based on the greatest common denominator of the processing times and then generating variables and constraints for these intervals. In this way, all the dynamics of the scheduling problem can be captured using the smallest formulation. Variables are generated within each interval to model the inventory levels ($a_{sjt'}$), task-unit assignments ($w_{ijt'}$), production levels ($b_{ijt'}$), and resource purchases ($qb_{st'}$). These variables are used to form constraint families that define feasible schedules. One important constraint family limits a unit to processing only one task at a time:

$$\sum_{i \in I_j} \sum_{x \mid (p_{ijt'} + x > t' \text{ and } x \le t')} w_{ijx} \le 1 \quad \forall j \in J \quad \forall t' \in T'. \quad (1)$$

A second constraint in the UDM formulation maintains a material balance from one time interval to the next for every resource. This constraint ensures that the amount of material in inventory at the end of the interval is equal to the amount in inventory at the beginning of the interval plus material arrivals due to purchases and completed tasks minus material consumed due to demands and tasks started during the interval:

$$\sum_{j \in J_s} a_{sjt'} = \sum_{j \in J_s} a_{sjt'-1} + qb_{st'} - d_{st'} + \sum_{i \in I} \sum_{j \in J_i} b_{ijt'-p_{ijt'}} f_{ijs}^{\text{create}} - \sum_{i \in I} \sum_{j \in J_i} b_{ijt'} f_{ijs}^{\text{consume}} \quad \forall s \in S \quad \forall t' \in T'. \quad (2)$$

The final important UDM constraint considered at this point limits the production capacity to be between a given upper and lower limit when a task is assigned to a unit:

$$V_{ij}^{\max} w_{ijt'} \ge b_{ijt'} \ge V_{ij}^{\min} w_{ijt'} \quad \forall i \in I \quad \forall j \in J_i \quad \forall t' \in T'. \quad (3)$$

The objective function utilized is the minimization of production costs (same as maximizing profit with fixed demands). Production costs include inventory ($C_{sjt'}^{\text{store}} a_{sjt'}$), raw material purchase ($C_{st'}^{\text{buy}} qb_{st'}$), as well as fixed ($C_{ijt'}^{\text{fixed}} w_{ijt'}$) and variable ($C_{ijt'}^{\text{variable}} b_{ijt'}$) task-unit operation costs. This gives us

$$\min \sum_{t' \in T'} \left(\sum_{s \in S} \left(\sum_{j \in J_s} C_{sjt'}^{\text{store}} a_{sjt'} + C_{st'}^{\text{buy}} qb_{st'} \right) + \sum_{i \in I} \sum_{J \in J_i} \left(C_{ijt'}^{\text{fixed}} w_{ijt'} + C_{ijt'}^{\text{variable}} b_{ijt'} \right) \right). \quad (4)$$

The major problem with this model, which was alluded to earlier, is that the greater the number of intervals, tasks, units, and/or resources, the larger the dimensionality of the formulation generated. Most importantly, the number of binary assignment variables increases, leading to greater combinatorial complexity. It is our present inability to effectively address the combinatorial complexity of the UDM formulation that is its greatest drawback, limiting its usefulness to smaller problems. However, the combinatorial complexity is an inherent feature of process-scheduling problems, and thus any solution methodology must address it in some manner.

Aggregate model formulation

From the UDM formulation, a more compact but less exact formulation can be easily constructed (Bassett et al., 1996). This formulation is referred to as an *aggregate* formulation because it is generated by summing over all the UDM variables and constraints for a predetermined set of discrete time intervals (in other words, aggregating the UDM formulation). This gives us

$$\min \sum_{t \in T} \left(\sum_{s \in S} \left(\sum_{j \in J_s} C_{sjt}^{\text{store}} A_{sjt} + C_{st}^{\text{buy}} QB_{st} \right) + \sum_{i \in I} \sum_{j \in J_i} \left(C_{ijt}^{\text{fixed}} W_{ijt} + C_{ijt}^{\text{variable}} B_{ijt} \right) \right) \quad (5)$$

$$\sum_{i \in I_j} p_{ijt} W_{ijt} \le H_t \qquad \forall j \in J \quad \forall t \in T \quad (6)$$

$$\sum_{j \in J_s} A_{sjt} + \sum_{i \in I} \sum_{j \in J_i} B_{ijt} f_{ijs}^{\text{consume}} + D_{st} = \sum_{j \in J_s} A_{sjt-1} + QB_{st} + \sum_{i \in I} \sum_{j \in J_i} B_{ijt-p_{ijt}} f_{ijs}^{\text{create}} \qquad \forall s \in S \quad \forall t \in T \quad (7)$$

$$V_{ij}^{\max} W_{ijt} \ge B_{ijt} \ge V_{ij}^{\min} W_{ijt} \qquad \forall i \in I \quad \forall j \in J_i \quad \forall t \in T, \quad (8)$$

where:

$$A_{sjt} = a_{sjt'} \qquad \forall s \in S \quad \forall j \in J_s \quad \forall t' = t \quad (9)$$

$$B_{ijt} = \sum_{t' \in t} b_{ijt'} \qquad \forall i \in I \quad \forall j \in J_i \quad (10)$$

$$D_{st} = \sum_{t' \in t} d_{st'} \qquad \forall s \in S \quad (11)$$

$$QB_{st} = \sum_{t' \in t} qb_{st'} \qquad \forall s \in S \quad (12)$$

$$W_{ijt} = \sum_{t' \in t} w_{ijt'} \qquad \forall i \in I \quad \forall j \in J_i \quad (13)$$

$$H_t = \sum_{t' \in t} 1 \qquad \forall t \in T. \quad (14)$$

Some assumptions are made when using the aggregate model formulation. The first is that any demands that are due during the time covered by an aggregate interval are assumed to be due at the end of the interval. This may artificially extend due dates that do not coincide with aggregate interval end points, so it is important to ensure that aggregation is done smartly to reduce this effect. In a similar manner, arrival and/or purchase of resources is allowed only at the beginning of aggregate intervals. Finally, due to the loss of sequencing characteristics, changeover costs and/or times are unable to be modeled within the aggregate formulation. This can lead to vastly overestimating production capacity (underestimating production costs) when changeover times (costs) play a significant roll in processing.

The value of this formulation, even with these assumptions, is that we are able to explore a longer horizon, but in far less detail than the UDM formulation. In the rest of this article, we present approaches that allow us to obtain the detail of the UDM formulation at the reduced expense of the aggregate formulation.

Time-based Decompositions

Current attempts to address the combinatorial complexity of process-scheduling problems have focused on time-based decomposition techniques. In this section, for completeness, we present a summary of the various time-based decomposition approaches we have investigated to date and discuss indications for their application. We begin by discussing a number of hierarchical approaches. In previous work, a hierarchy of product families and types has been used to reduce the problem size (Bitran and Hax, 1977; Bitran et al., 1981). We present a hierarchy that separates the process-scheduling problem into a planning level and scheduling level. Under such approaches the planning and scheduling problems are solved iteratively, but the various approaches utilize different techniques to remove infeasibilities. The first approach is a heuristic technique presented in Bassett et al. (1996) which utilized an *ad-hoc* modification of infeasible schedules to obtain feasible solutions. Subsequent approaches use constraint modification and constraint disaggregation to remove infeasible solutions from consideration, thus removing the *ad-hoc* nature of the first approach.

Based on the discussion of the hierarchical approaches, a rolling-window approach is presented. This approach utilizes a hybrid planning/scheduling formulation that removes the necessity of employing a hierarchical update scheme. Finally, a new heuristic is presented that further reduces the complexity of the rolling window hybrid formulation by utilizing information from solutions obtained in earlier stages of the algorithm.

Hierarchical approaches

Using the formulations discussed in the previous section, the first approach taken was to extend the hierarchical decomposition approach by Subrahmanyam et al. (1994). Although this hierarchical approach was initially proposed for the separation of design and scheduling decisions, in Bassett et al. (1996) the approach was extended by introducing a planning layer between the design and scheduling layers. The key to all of these hierarchical decomposition methods lies in how the interaction between the planning and scheduling layers is accommodated. For more details on the design layer see Subrahmanyam et al. (1994) and Bassett et al. (1996).

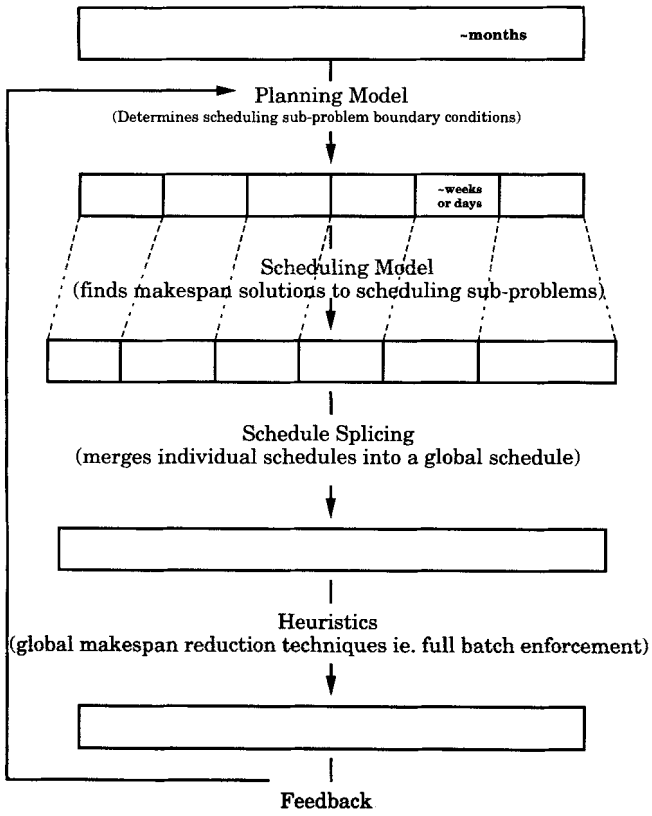

Figure 2. Local schedule adjustment hierarchical approach.

Local Schedule Adjustment. In Bassett et al. (1996), an heuristic approach was presented to achieve feasible solutions to the overall scheduling problem. The progression of the method is shown in Figure 2. Initially, a planning problem is constructed that divides the horizon into a number of aggregate intervals. The solution of this problem allocates production quantities within each aggregate interval. Next, scheduling subproblems are generated for each aggregate interval and solved independently. If any scheduling subproblem is infeasible (takes longer to complete production than the length of the aggregate interval), a series of heuristics are

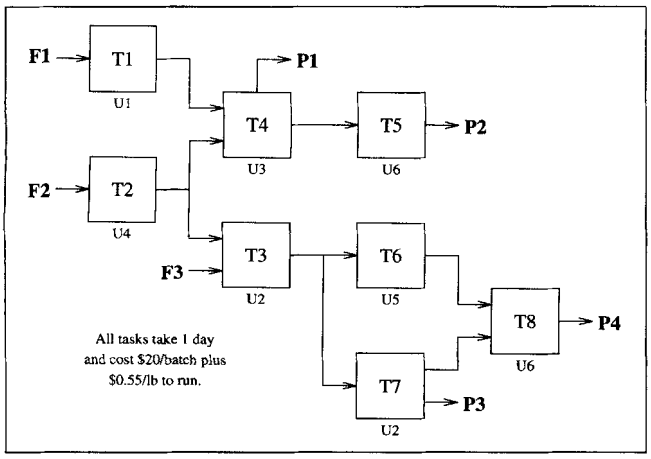

Figure 3. Recipe for Extended__BATCH4.

Table 1. Demands for Extended__BATCH4

Aggregate Interval	Day of Demand	P1	P2	P3	P4
1	2	110	0	0	0
1	3	110	233.1	0	0
1	4	133.3	260	116.6	0
1	5	100	0	56.6	333.3
1	6	33.3	360	0	333.3
1	7	33.3	360	116.6	685.8
2	14	3,000	4,000	0	0
3	21	0	5,500	0	0
4	28	2,500	0	1,300	3,000
5	56	12,000	1,000	1,000	1,000
6	84	1,000	1,000	7,000	10,000
7	112	10,000	16,000	2,000	2,000

applied to hopefully remove this infeasibility. These heuristics attempt to reduce idle time by shifting batches and/or merging partial batches. If these heuristics are unable to achieve a feasible solution, the planning problem has to be modified to reduce available production time and/or capacity and the process repeated.

This approach was applied successfully to two example problems. The first problem was a variation of BATCH4 from Sahinidis and Grossmann (1991) that we will call Extended__BATCH4. This example was chosen because it is large enough to test the approach yet small enough to be solvable without decomposition. The recipe for producing the four final products is shown in Figure 3. Based on the demands given in Table 1, the planning model was formulated using a nonuniform discretization of the horizon. This resulted in having to consider only seven time intervals within the planning horizon. The demands for the first week of production are aggregated into single demands for each product since the frequency of those demands was less than the production time for the products. In Table 2, a comparison of both solution value and time is given for solving the whole problem directly vs. utilizing hierarchical decomposition.

Even though we are only attempting to get a feasible solution with the hierarchical decomposition approach, the solution obtained is within 1% of the value obtained by solving the undecomposed problem. It should be noted that none of the heuristics were needed to obtain the solution: that is, feasible schedules were obtained for all aggregate intervals. The solution times for both approaches are on the same order of magnitude even though more problems are solved in

Table 2. Results for Extended__BATCH4

Approach	Objective Value	Solution Time (cpu s)*
Global Solution	**528,324.3604**	**21.54**
Planning model solution		6.50
Time interval 1	6,217.4557	0.39
Time interval 2	14,660.0000	0.19
Time interval 3	9,127.8625	0.18
Time interval 4	19,982.3728	0.30
Time interval 5	97,750.0000	0.92
Time interval 6	172,146.7310	1.47
Time interval 7	211,866.8752	1.19
Total	**531,751.2972**	**11.13**

*Solved on HP 9000/735.

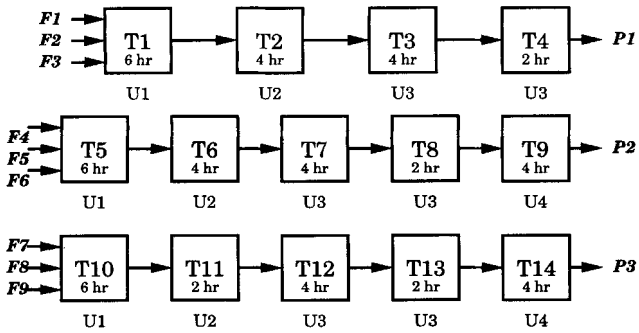

Figure 4. Recipe of multisite scheduling problem.

the decomposition approach. The important point is that the smaller scheduling subproblems are easier to solve than the global scheduling problem, since as the overall problem scales in size the solution time becomes prohibitive.

To test the approach on a more difficult problem, a multisite scheduling problem was constructed based on industrial data. The problem consists of two plants producing three products over one and a half months (discretized to 918 hours) to supply two warehouses. Each product takes roughly five tasks on four units to be produced, with two products being produced in one plant and all three in the other. See Figures 4 and 5 for the recipe and transportation tasks.

To solve the problem, the horizon was divided into nine equal aggregate intervals (102 hours each) with all the demands due in the final interval (see Table 3). The solution for the multisite planning problem gave the production quantities of each product in each plant for all nine intervals. Since the plants operate independently (no shipping of intermediates between plants), the scheduling problems for each plant can be solved independently, giving eighteen scheduling problems. In this case, numerous scheduling problems proved to be infeasible. Looking at two adjacent time intervals for plant one (Figures 6 and 7), it can be seen that end effects play a major role in causing infeasibilities. At the beginning

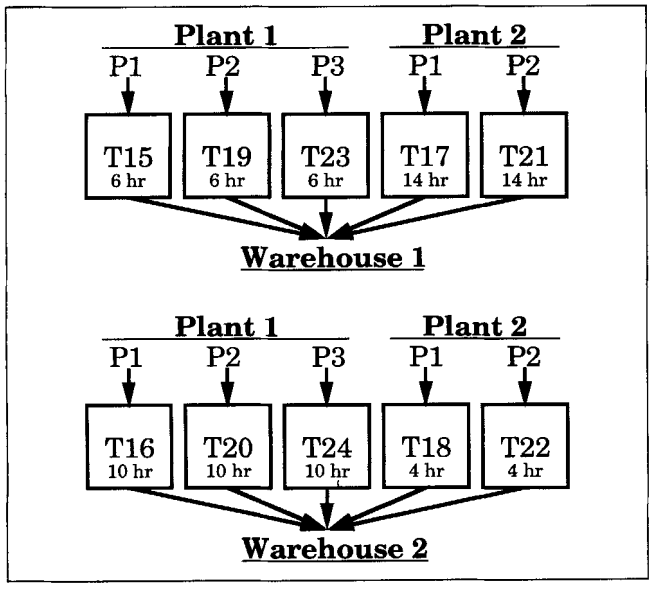

Figure 5. Transportation tasks for multisite scheduling problem.

Table 3. Demands for Multisite Scheduling Problem

Demand Distribution	Quantity	Hour Due	Aggregate Interval
P1 to warehouse 1	120,000	918	9
P1 to warehouse 2	60,000	918	9
P2 to warehouse 1	50,000	918	9
P2 to warehouse 2	40,000	918	9
P3 to warehouse 1	20,000	918	9
P3 to warehouse 2	70,000	918	9

and end of each time period shown, all equipment is forced to be idle, thereby losing valuable production time and capacity due to an artificially induced end effect. Through both internal shifting of tasks (arrows in figures) and splicing the end of one interval to the beginning of the next, a significant reduction in the infeasibility of the global solution is obtained as seen in Figure 8. However, even splicing and shifting are unable to remove all the infeasibility, and partial batch shifting/merging is needed to obtain the feasible solution shown in Figure 9. For a more detailed discussion of the heuristics used, see Bassett et al. (1996).

Merging, shifting, and splicing of the solutions significantly change the inventory levels, batch sizes, etc., requiring post-processing to determine the final values. It is the *ad-hoc* manner in which infeasibilities are removed that is the greatest drawback of the approach presented. However, based on the success of the two examples we decided to pursue an algorithmic approach based on hierarchical decomposition that used a more systematic means of combining subproblem information.

Constraint Modification. To remove the *ad-hoc* nature of the previous approach, the splicing, shifting, and merging heuristics were replaced. Instead, two sets of constraints were introduced to modify the planning problem based on infeasible scheduling-subproblem solutions.

Using the UDM formulation, if a scheduling problem is infeasible, very little information is obtained unless a detailed analysis of the slack variables is undertaken. However, there is a simpler way to determine the nature of the infeasibility by the introduction of excess batch variables ($e_{ijt'}$) and intermediate purchase variables ($qb_{st'}$ $\forall s \in S^{\text{intermediate}}$). These variables, in essence, replace the slack variables introduced by the LP solver when they are added to the material balances (Eq. 2)

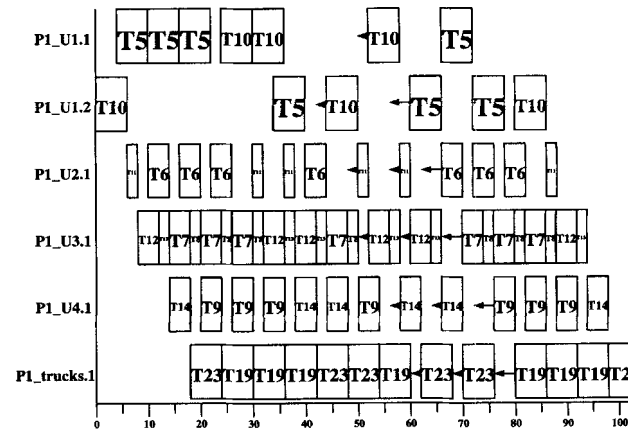

Figure 6. Multisite time period 1.

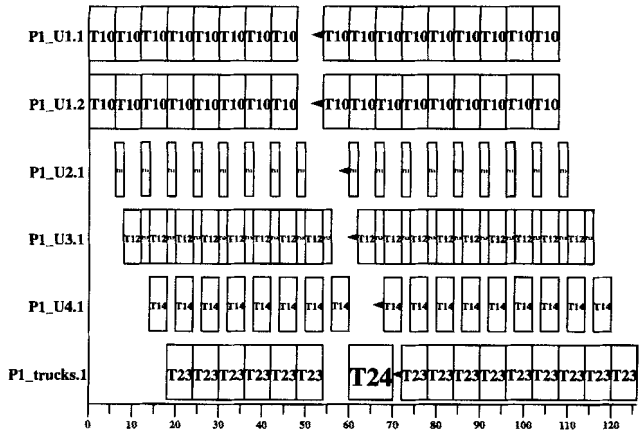

Figure 7. Multisite time period 2.

$$\sum_{j \in J_s} a_{sjt'} + \sum_{i \in I} \sum_{j \in J_i} (b_{ijt'} + e_{ijt'}) f_{ijs}^{\text{consume}} + d_{st'} = \sum_{j \in J_s} a_{sjt'-1} + qb_{st'} + \sum_{i \in I} \sum_{j \in J_i} (b_{ijt'-p_{ijt'}} + e_{ijt'-p_{ijt'}}) f_{ijs}^{\text{create}} \quad \forall s \in S \quad \forall t' \in T' \quad (15)$$

and replace the objective function (Eq. 4) with

$$\text{Minimize} \sum_{t' \in T'} \left(\sum_{i \in I} \sum_{j \in J_i} C_{ijt'}^{\text{variable}} e_{ijt'} + \sum_{s \in S^{\text{intermediate}}} C_{st'} qb_{st'} \right). \quad (16)$$

By solving the modified scheduling subproblem, any infeasibility due to capacity limitations or intermediate resource availability will result in a nonzero objective value. Based on the values obtained for the excess batch and intermediate purchase variables, two sets of constraints may be added to the planning formulation. The first set of constraints is introduced if any excess batch variable is nonzero. A nonzero $e_{ijt'}$ simply means that task i on unit j did not have enough capacity at time t' to meet its production requirements. This implies that the planning problem overestimated the plant's capacity for task i on unit j in aggregate interval t and needs to be updated to account for this error. An upper limit on the aggregate production capacity for a task-unit pair with nonzero excess batch variables is given by

$$B_{ijt} \le \sum_{t' \in t} b_{ijt'} \Big| \sum_{t' \in t} e_{ijt'} > 0 \quad \forall i \in I \quad \forall j \in J_i \quad \forall t \in T. \quad (17)$$

These are called *task-unit aggregate batch limitation* (TUABL) *contraints*. In addition, constraints can be added based on a task, unit, and/or group (a set of identical units) basis. This leads to *task aggregate batch limitation* (TABL) *constraints*

$$\sum_{j \in J_i} B_{ijt} \le \sum_{j \in J_i} \sum_{t' \in t} b_{ijt'} \Big| \sum_{j \in J_i} \sum_{t' \in t} e_{ijt'} > 0 \quad \forall i \in I \quad \forall t \in T, \quad (18)$$

unit aggregate batch limitation (UABL) *constraints*

$$\sum_{i \in I_j} B_{ijt} \le \sum_{i \in I_j} \sum_{t' \in t} b_{ijt'} \Big| \sum_{i \in I_j} \sum_{t' \in t} e_{ijt'} > 0 \quad \forall j \in J \quad \forall t \in T, \quad (19)$$

and *group aggregate batch limitation* (GABL) *constraints*

$$\sum_{j \in \text{Group}} \sum_{i \in I_j} B_{ijt} \le \sum_{j \in \text{Group}} \sum_{i \in I_j} \sum_{t' \in t} b_{ijt'} \Big| \sum_{j \in \text{Group}} \sum_{i \in I_j} \sum_{t' \in t} e_{ijt'} > 0 \quad \forall \text{ Group} \quad \forall t \in T. \quad (20)$$

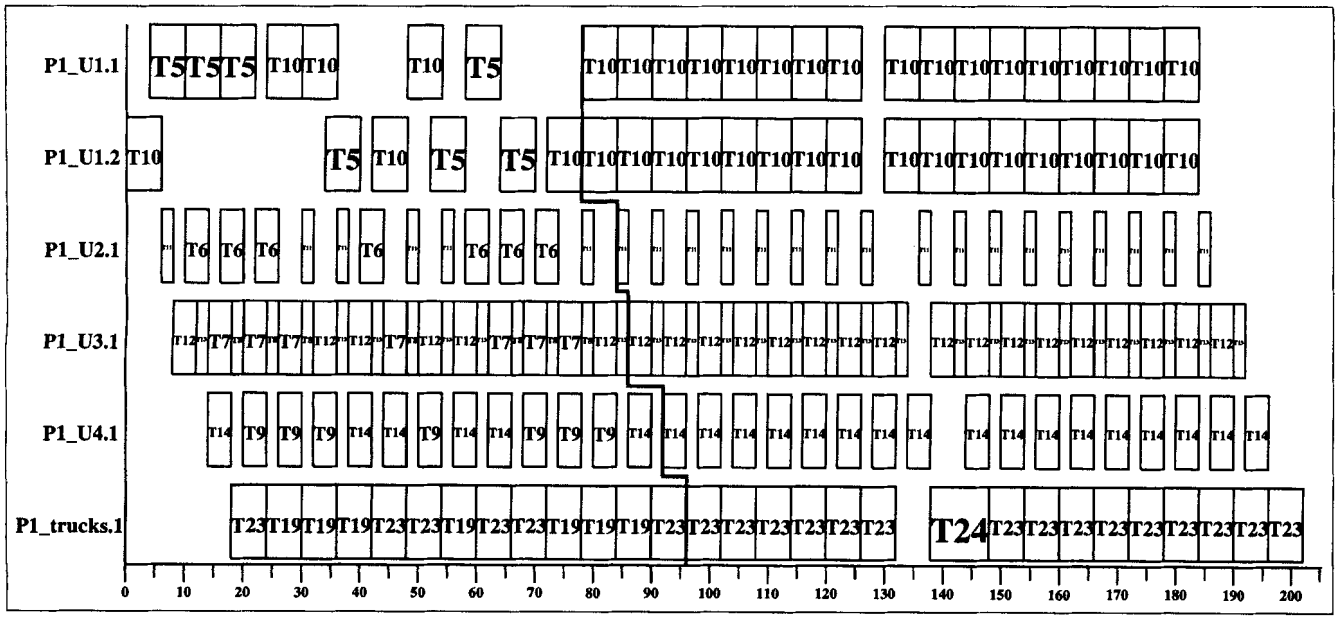

Figure 8. Multisite time periods 1 and 2 spliced.

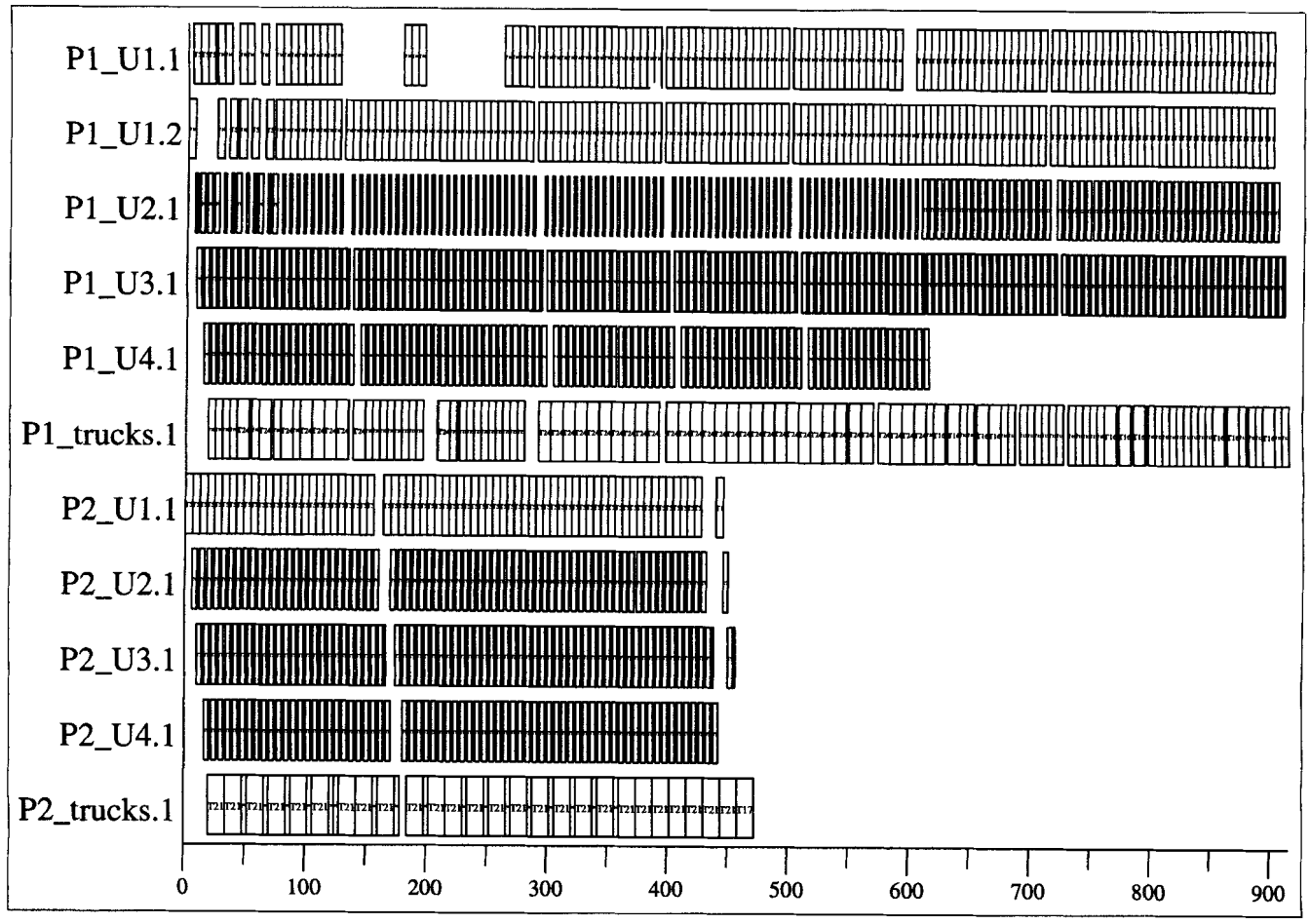

Figure 9. Global solution to multisite example.

One can think of the previous constraints as crude cutting planes that remove previous solutions from future consideration by limiting production within the aggregate intervals of the planning problem.

The second set of constraints is added if any intermediate purchase variables are nonzero. The *intermediate target inventory* (ITI) *constraints*

$$A_{st} \geq \sum_{t' \in t} qb_{st'} \qquad \forall s \in S^{\text{intermediate}} \quad \forall t \in T \qquad (21)$$

ensure that future solutions to the planning problem will have enough of the previously limiting intermediates on hand at the right time so that production is not affected.

By adding these two sets of constraints, we remove the previous aggregate solution from consideration and thereby obtain a different production plan. The algorithm iterates between solving the planning problem, solving the scheduling subproblems, and updating the planning problem until all scheduling subproblems are feasible or the planning problem is infeasible.

During the implementation of this approach, a number of problems were encountered that limit the usefulness of the algorithm. First, artificial end effects were not always removed by the addition of these constraints. These effects are introduced by the aggregate intervals of the planning problem that require equipment to be idle at their beginning and end. Artificial end effects are a problem in two obvious cases: if production for a given demand occurs in more than one aggregate interval and/or if cycle times between storeable intermediates are on the order of aggregate interval lengths. For both these cases, artificial breaks in the schedules lead to underutilization of the available plant capacity. Second, the number of iterations needed for the algorithm to terminate is unknown, possibly leading to unreasonably long solution times. However, since the scheduling subproblems are independent, this method is amenable to parallel implementation as discussed in Subrahmanyam et al. (1996a).

Even with its limitations, this approach is still useful for a number of interesting problems. If a large number of resources are storable or the production chain is short, the process cycle time is reduced to the time between storable resource. This minimizes the end effects artificially introduced by the algorithm. Also, if the scheduling problem is not very busy (light demands, excess production capacity) such that production does not overlap aggregate intervals, the production underestimation introduced by artificial end effects does not lead to infeasibilities. For instance, the solution for the problem Extended_BATCH4, presented in the subsection titled "Local Schedule Adjustment," is easily obtained by means of this algorithm.

The next approach overcomes the problem of end effects as well as that of excessive numbers of iterations.

Contraint Disaggregation. End effects appear in the scheduling subproblems because each of the aggregate intervals is treated independently of its neighbors. Therefore, tasks are not allowed to begin in one aggregate interval and then finish in the next aggregate interval. In other words, the initial and final state of each aggregate interval must have all equipment idle. It is obvious that this will severely limit the capacity of the plant in a manner that is highly dependent on the aggregate interval lengths.

In order to take care of these effects the disaggregation approach updates the planning problem by disaggregating the aggregate constraints within the planning problem for all infeasible scheduling subproblems. In this way, all the infeasibility for those intervals is shifted to neighboring intervals that are possibly less constrained. The obvious problem with this approach is that we are forced to solve a more difficult hybrid planning/scheduling problem which, in the extreme, approaches the global UDM formulation of the problem at hand. This algorithm is assured to only take, at most, as many iterations as there are aggregate intervals. But since the solution of the hybrid formulation at each iteration is progressively harder, this reduction in iterations is useful when only a few aggregate intervals need expansion. A detailed discussion of optimality conditions and theoretical bounds for this approach is given in Subrahmanyam et al. (1996b).

In considering this approach a number of important features became apparent. In particular, consider a problem with a single demand due at the end of the horizon. First, it was found that if any interval was infeasible, the aggregate interval immediately adjacent to the demand was the most infeasible. This is reasonable if one remembers that the objective is to minimize production costs, of which inventory costs play a large role. This forces production to be pushed as close to when the demand is due as possible, but since the aggregate formulation overestimates the production capacity, the UDM formulation for this interval is usually infeasible. Disaggregation of this interval in the planning problem maximizes production as close to the demand as possible while shifting any infeasibilities to the next earlier aggregate interval. Now this adjacent interval is tested as before, and so on, until either the hybrid planning/scheduling problem is infeasible or all the scheduling subproblems are feasible.

As illustrated by the progression of disaggregations for a worst-case example, shown in Figure 10, the feasibility-seeking process could behave like a reverse rolling window. Thus, although the scheme has been proven to converge to the optimum for some problems (Subrahmanyam et al., 1996b), in the worst case it could require solving the entire disaggregation model. The next section discusses a rolling-windowapproach that streamlines this approach by not enforcing
optimality.

Reverse rolling window

The crude rolling window achieved while implementing the planning formulation disaggregation approach can be significantly streamlined by removing the intermediate scheduling subproblems. This intermediate step is unnecessary, since by introducing the disaggregate constraints into the hybrid formulation we are already solving the scheduling subproblem. If we fix the solution obtained for each previously disaggregated interval, then the problem solved at each step is only slightly larger than that of a scheduling subproblem.

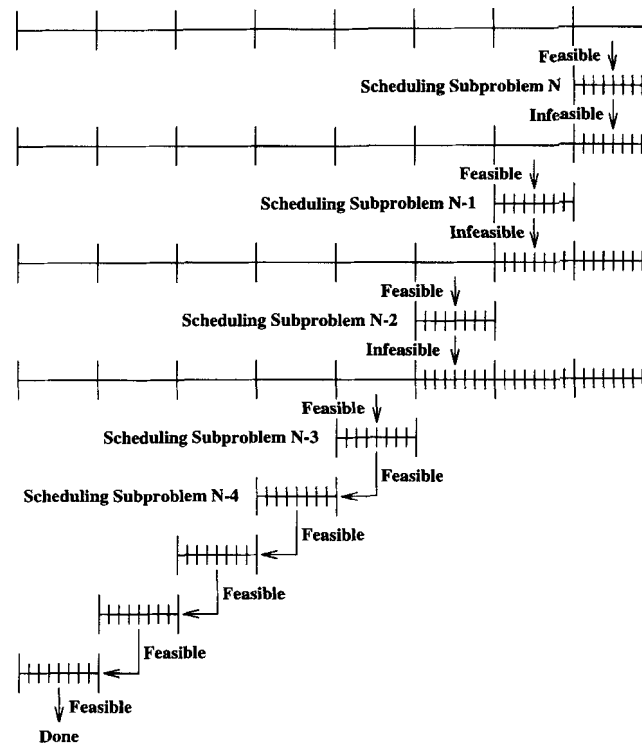

Figure 10. Progression of worst-case example.

However, in fixing the solution of previous windows, we are assuming that the locally obtained schedule is globally feasible. This is reasonable for large windows and is, of course, correct if the window is equal to the length of the horizon. But the whole reason for exploring this approach is to reduce the size of the UDM formulation that must be dealt with at any one time. However, for small windows, fixing the solution obtained may lead to locally infeasible solutions for problems that actually have globally feasible solutions. To overcome this problem, two improvements to the rolling window have been introduced.

Reverse Rolling Window with Crossover. The first modification is based on an idea introduced by Wilkinson et al. (1994), namely, that of adding crossover variables to connect aggregate intervals within a planning problem. Crossover variables are UDM variables that allow tasks to begin in one aggregate interval and finish in the next aggregate interval (correspond to additional $w_{ijt'}$ and $b_{ijt'}$ variables). This removes the requirement that all equipment is idle at the beginning and end of every interval, thereby negating end effects. In our rolling window, the only place where end effects play a role are the point at which the UDM and aggregate formulation meet, so crossover variables are introduced at this point. This modification only slightly increases the problem size due to the increased number of variables and constraints. In essence, for a given window size, a formulation slightly larger than the window is generated. As the algorithm progresses, only by fixing the variables contained in the original window, modifications

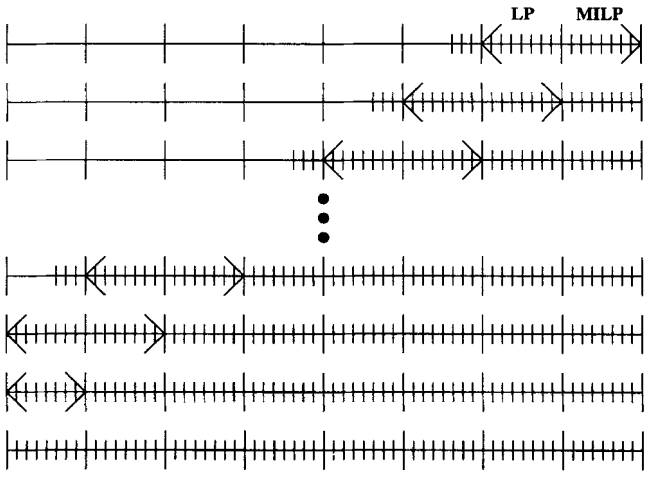

Figure 11. Reverse rolling-window with crossover variables and LP region.

can be made to the crossover variable values, allowing the adjacent window to mesh with the old.

Reverse Rolling Window with Crossover and LP-MILP Regions. The second modification to the rolling-window approach further exploits the idea of allowing adjacent windows to have a chance to mesh together. The crossover variables introduce meshing, but sometimes this is simply not enough to overcome the infeasibilities due to locally determined feasible solutions. As discussed earlier, we could further increase the window size to reduce this infeasibility, but this can be carried only so far before the combinatorial complexity, due to the binary assignment variables ($w_{ijt'}$), makes the formulation solution intractable. If we only enforced the binary nature of the assignment variables for a smaller subwindow, we could reduce the combinatorial complexity of the problem. This gives us both an LP and MILP region within the rolling window.

Of course, within the LP region we no longer obtain a feasible schedule since the integrality of the assignment variables is not enforced. However, we still retain the detailed nature of the material balances that maintain the precedence relationships among tasks. So while the model employed over the LP region is not exact, it is a more faithful representation than the aggregate model. At some point, however, the binary nature of the assignment variables must be enforced. The easiest manner to do this is to only shift the rolling window by the width of the MILP subwindow. This places the MILP subwindow over all or part of the LP subwindow, thus enforcing the binary nature of the assignment variables in this region (see Figure 11).

As an initial test of the rolling-window approach, Extended BATCH4 was solved. For comparison with the reverse rolling window, a forward rolling window similar to Wilkinson et al. (1994) was also implemented. Their rolling window did not contain an LP region so we did not introduce this region for this comparison. As seen in Table 4, the reverse rolling window is able to obtain a feasible solution to the problem with a window four times smaller than the window needed by the forward rolling window. The size window needed by the forward-rolling-window approach is of the same order of magnitude as the widest spaced demand for products (see Table 1)

Table 4. Comparison of Forward and Reverse Windows without LP Region

Min. MILP Window Needed		Subproblems Solved		Solution Time (cpu s)*	
Forward	Reverse	Forward	Reverse	Forward	Reverse
28	7	4	16	7 s	11 s

*Includes time to generate subproblems and solve them on HP 9000/735.

while the reverse-rolling-window size is on the order of the longest processing time for the tasks (see recipe in Figure 3).

This disparity between the window sizes can be explained by much the same argument used to introduce the reverse rolling window. The objective of the scheduling problem is to minimize production costs, of which inventory costs play an important role, so as much production as possible is pushed as close to when demands are due. Since the aggregate formulation overestimates the production capacity, infeasibilities appear near demands. With the reverse rolling window, the window starts at the latest demand and pushes any infeasibilities in front of it (into earlier, hopefully less busy, intervals). This means the reverse window only needs to be large enough to take care of precedence relationships within the recipe. With the forward rolling window, however, the window needs to be large enough such that it can anticipate future demands and start production early enough to avoid the infeasibilities close to the demands. For this example, only a window as large as the widest spaced demand was able to achieve this.

The penalty that comes with having a smaller window, however, is having to formulate and solve more subproblems. It is this reformulation cost that leads to the higher solution time for the smaller window. For larger problems, the cost to reformulate plays a much smaller role in the overall solution time, and the savings due to solving smaller hybrid formulations, is evident as shown in the following discussion.

Next, we allowed the addition of an LP region to both rolling windows to see what effect it would have on Extended BATCH4. Table 5 shows the minimum LP region subwindow size for a given MILP region subwindow size. For the forward-rolling-window approach, the smaller the MILP region, the larger the necessary LP region needed to achieve a feasible solution. For the case of a MILP window of size seven, the forward approach is unable to achieve a feasible solution for any size LP region. For the reverse case, an LP region is unnecessary for any size MILP region studied.

The cost of having to solve a larger subproblem at each iteration is clearly evident in comparing the solution times. When the size and number of subproblems solved by each

Table 5. Comparison of Forward and Reverse Windows with LP Region

MILP Width	Min. LP Width Needed		Subproblems Solved	Solution Time (cpu s)*	
	Forward	Reverse		Forward	Reverse
7	—	0	16	—	11 s
14	56	0	8	20 s	7 s
28	0	0	4	7 s	7 s

*Includes time to generate subproblems and solve them on HP 9000/735.

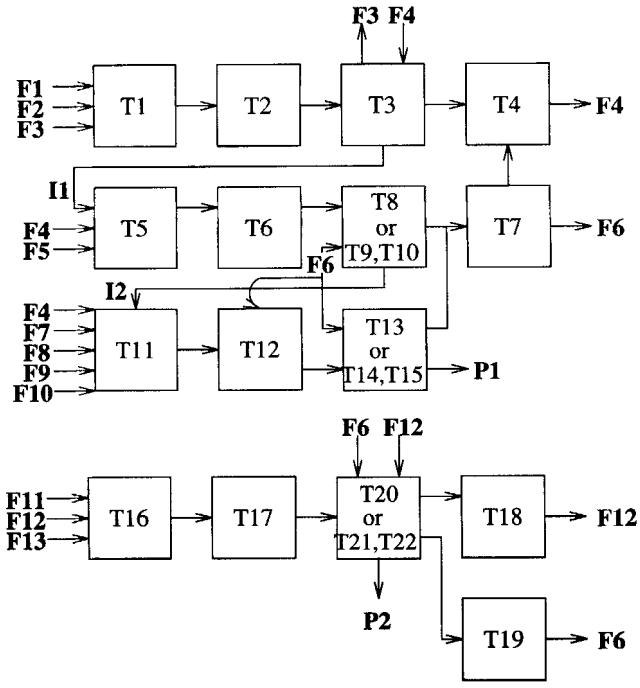

Figure 12. Recipes for P1 and P2 of industrial case study.

approach are the same, the overall solution time is the same. As the LP region increases in size, the overall solution time increases as well. This is seen when comparing the two cases where the MILP interval width is fourteen.

Next, we looked at an industrial case study to see how this approach works on a large-scale problem. The recipes, possible task-unit assignments, and processing times are given in Figure 12 and Table 6. The problem entails producing two products using 22 tasks on 12 pieces of equipment utilizing 31 resources. The horizon of three months has been discretized into 1,800 intervals. Each product has a single demand due at the end of the horizon: P1—100,000 and P2—95,000.

For this example, only a reverse rolling window was used since the widely spaced demands lead to a forward rolling window of enormous dimensions, as discussed in the previous examples. For the reverse rolling window, an MILP window of 20 intervals was chosen since the longest processing time in the recipe was 18 intervals. Using this width, a total of 90 subproblems were solved and a feasible solution was obtained, without the need for an LP region, in approximately 8.25 cpu hours. We are unable to compare this solution time to the time needed to solve the global problem due to the fact that the global formulation would contain over 100,000 binary variables plus the continuous variables. The schedule

Table 6. Task-Unit Assignments and Processing Times for Industrial Case Study

Units	Tasks (Processing Times)
U1, U2, U3	T1(12), T5(12), T11(12), T16(6)
U4, U5, U6	T2(3), T6(3), T12(2), T17(2)
U7, U8, U9	T3(7), T8(11), T9(7), T13(8), T14(4), T20(18), T21(11)
U10	T10(3), T15(3), T22(5)
U11, U12	T4(6), T7(6), T18(6), T19(6)

Table 7. Task-Unit Assignments for Industrial Case Study

Unit	Task (Number of Times Performed)
U1	T1(30), T5(20), T11(33), T16(25)
U2	T1(32), T5(23), T11(24), T16(36)
U3	T1(27), T5(18), T11(39), T16(32)
U4	T2(24), T6(21), T12(40), T17(30)
U5	T2(37), T6(23), T12(28), T17(31)
U6	T2(27), T6(17), T12(28), T17(29)
U7	T3(27), T8(7), T9(16), T13(2), T14(32), T20(8), T21(23)
U8	T3(33), T8(5), T9(14), T13(2), T14(27), T20(6), T21(28)
U9	T3(28), T8(3), T9(16), T13(3), T14(30), T20(14), T21(14)
U10	T10(47), T15(89), T22(65)
U11	T4(73), T7(83), T18(24), T19(0)
U12	T4(71), T7(81), T18(27), T19(0)

obtained by the reverse-rolling-window approach schedules 1,570 tasks. Since the Gantt Chart for a problem of this magnitude is impossible to view in detail without a zoomable Gantt Chart, Table 7 shows how many times each task was performed on each unit. This helps give an idea of the complexity of the schedule obtained.

Even though 1,570 task-unit assignments are scheduled, 99,674 other task-units assignments were not (98.5% of $w_{ijt'} = 0$). This means a given subproblem contains $\sim 1{,}100$ binary variables of which only ~ 18 are actually used. This observation provides a further opportunity to reduce execution time, as we now discuss.

Disaggregation Heuristic. The reason for the disparity between the number of variables generated and the number of tasks executed can be attributed to two major factors: equipment sharing and precedence relationships. When more than one task can be performed on a piece of equipment, the assignment constraints assure that only one task is performed at a time. Therefore, if five tasks may be performed on one unit, as soon as one task is selected, the other four are zero. In addition, for the duration of the time spent processing the selected task, all other assignment variables are zero as well. The second factor is due to precedence relationships. If a resource must be available for a task to run, the task producing the resource must have run prior to this task. Thus, if task B needs a resource from task A and if task A is not run, then task B cannot be run as well.

We would like to determine *a priori* which task-unit combinations have a better chance of being assigned than others. One method for doing this is to utilize the solution from the previous subproblem to guide the generation of the next subproblem. When we solve a hybrid formulation, we not only obtain an exact solution for the MILP region, we get a "best guess" solution for the LP region as well as the aggregate region. The approach described earlier utilizes the information from the MILP region, but the other regions can give us important information too.

From the aggregate region we can look at the W_{ijt} variable values to give us guidance. If $\Sigma_{j \in J_i} W_{ijt} = 0$, we know that there is a very good probability that the assignment variables for this task in the UDM formulation will be zero as well. We can base this assumption on the fact that the aggregate formulation overestimates the production capacity and, as such, it is highly unlikely for the detailed UDM formulation to perform tasks not performed in the aggregate formulation. A similar argument can be used to reduce the variables gener-

Table 8. Task-Unit Assignments for Industrial Case Study with Disaggregation Heuristic

Unit	Task (Number of Times Performed)
U1	T1(28), T5(20), T11(32), T16(16)
U2	T1(30), T5(20), T11(33), T16(30)
U3	T1(31), T5(20), T11(31), T16(47)
U4	T2(23), T6(15), T12(37), T17(13)
U5	T2(12), T6(16), T12(32), T17(52)
U6	T2(24), T6(29), T12(27), T17(28)
U7	T3(15), T8(0), T9(22), T13(0), T14(33), T20(0), T21(31)
U8	T3(16), T8(0), T9(21), T13(0), T14(33), T20(0), T21(31)
U9	T3(28), T8(0), T9(17), T13(0), T14(30), T20(0), T21(31)
U10	T10(62), T15(96), T22(93)
U11	T4(61), T7(77), T18(38), T19(0)
U12	T4(69), T7(73), T18(47), T19(0)

ated based on the $w_{ijt'}$ variable values from the LP region. As an aside, this approach could be used in a delayed column generation scheme, under which only columns that correspond to likely assignment variables are generated as needed.

When the disaggregation heuristic is applied to the industrial case study just solved, the solution time was decreased from 8.25 cpu hours to 23 cpu minutes (a 95% decrease in solution time), making the reverse rolling window a viable approach. This decrease was accomplished through a reduction in the number of assignment variables from 98,244 to 32,268 (a 67% decrease). A total of 1,570 tasks were still scheduled (see Table 8), although the task-unit assignments vary slightly from the previous solution.

To verify the viability of the reverse rolling window with the disaggregation heuristic, a number of random problems were generated using a uniform distribution. Each of these problems uses the same recipe as the previous example while allowing the processing times to be -1, 0, or 1 interval widths different from the base times shown in Table 6 (rounding was used to obtain integer processing times). This corresponds to varying the processing times from $\pm 5\%$ to $\pm 50\%$. In addition, the demands were allowed to be due in 1, 2, or 3 evenly spaced intervals in addition to varying $\pm 10\%$. For each demand spacing, five random problems were generated with their specifics, as shown in Table 9. The actual processing times used for each problem are not shown due to space limitations.

Table 9. Demands for Randomly Generated Problems

Problem	Interval	P1 Demand	P2 Demand
1	1,800	86,783	91,303
2	1,800	88,944	103,551
3	1,800	104,493	93,215
4	1,800	101,042	102,880
5	1,800	91,978	109,960
6	900	50,792	45,497
	1,800	52,202	50,944
7	900	50,353	50,330
	1,800	50,892	49,128
8	900	42,899	46,454
	1,800	45,160	50,265
9	900	44,945	52,578
	1,800	48,927	51,403
10	900	49,036	54,826
	1,800	46,962	53,678
11	600	29,348	30,197
	1,200	34,694	34,122
	1,800	33,372	35,020
12	600	33,820	30,677
	1,200	33,073	35,652
	1,800	34,819	34,712
13	600	29,999	34,759
	1,200	32,849	36,411
	1,800	32,589	36,148
14	600	32,074	30,453
	1,200	32,474	33,230
	1,800	33,095	34,096
15	600	30,763	31,952
	1,200	32,025	34,747
	1,800	28,635	30,300

Table 10. Results for Randomly Generated Problems

Problem	Assignment Variables	Assignments Made	Solution Time (cpu s)*
1	32,772	1,455	1,378.66
2	47,920	1,559	2,341.87
3	35,010	1,672	1,641.66
4	35,591	1,626	2,196.15
5	50,335	1,665	1,965.77
6	48,821	1,657	1,568.52
7	47,041	1,680	1,977.90
8	46,442	1,550	2,436.17
9	40,943	1,544	2,444.56
10	37,565	1,695	1,697.06
11	43,622	1,624	1,667.33**
12	59,022	1,698	2,266.96
13	45,068	1,664	2,111.02
14	49,302	1,620	1,858.61
15	38,471	1,545	1,866.19

*Solved on HP 9000/735.
**Reached 1,500-node limit for one subproblem.

All problems were solved with an optimality tolerance of 0.00125 or a maximum of 1,500 nodes for each branch and bound tree. An MILP window of 20 intervals with no LP region was used, giving a total of 90 subproblems to be solved for each problem. Feasible solutions were obtained for all 15 random problems, with all the solutions taking between 22 and 40 minutes (see Table 10). The total number of assignment variables generated for all the problems is well below the $\sim 100{,}000$ that would be generated for the global scheduling problem. The actual number of assignments made, obviously, scales with the demand quantities; thus some problems require more tasks to be performed than others. Based on these results we see that for this recipe, solution time is not significantly affected by changes in the processing times, demand levels, or demand spacing.

A possible problem with the disaggregation heuristic, but which has not been encountered as yet in our testing, is that it may be overly pessimistic about which tasks will be performed, thereby removing too many variables from consideration, leading to infeasible solutions. If this happens, the only way to currently overcome this problem is to start over with a less aggressive heuristic or without using a heuristic at all. As we saw from the industrial case study, without any heuristics the solution times become unacceptable.

It would be useful to have a way to determine which variables need to be added to the hybrid formulation to achieve feasibility without having to resolve the entire problem from the beginning. In order to determine the nature of the infeasibility, variables can be introduced into the hybrid formula-

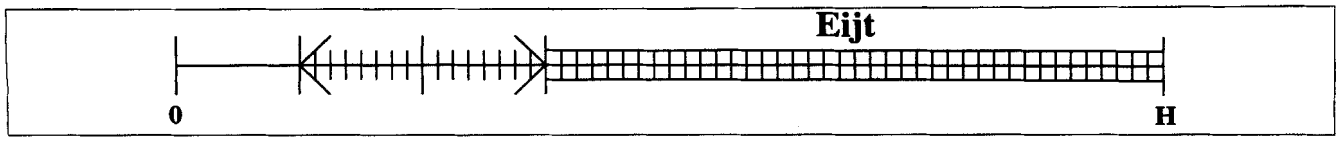

Figure 13. Introduction to excess batch capacity variables.

tion in a manner similar to the constraint modification approach presented earlier.

As the window moves and leaves behind a fixed solution, this solution is aggregated into a single interval with the variable bounds being determined by the solution. When an infeasibility occurs, we can calculate what needs to be done by introducing aggregate excess batch variables (E_{ijt}) into the aggregate constraints for the fixed region and resolving the problem with the objective of minimizing the excess variable values (see Figure 13). If $E_{ijt} > 0$ for a task that has not been disaggregated in the current window, disaggregate and resolve. If a feasible solution is then achieved, continue the algorithm as before. If the problem is still infeasible or $E_{ijt} > 0$ for an already disaggregated task, back the window up one step and attempt to disaggregate. At some point either an interval will be found that has not already disaggregated this task or the window will move to the horizon, that is, the problem is infeasible. Moving the window back in time is not a trivial component of the heuristic. There are many issues to be explored, not the least of which is how to retain previous subproblems without overloading memory.

Other Decompositions/Aggregations

So far, all the approaches presented attempt to reduce problem complexity through a time-based decomposition and task-unit aggregation ($W_{ijt} = \Sigma_{t' \in t} w_{ijt'}$). As can be seen in the complexity cube of Figure 1, this is only one component of the overall complexity. Approaches that decompose and/or aggregate tasks/resources ($W_{Ijt'} = \Sigma_{i \in I} w_{ijt'}$) and/or units ($W_{iJt'} = \Sigma_{j \in J} w_{ijt'}$) should also help reduce problem complexity.

Resource decomposition

Resource decomposition can be used to reduce the task/resource component of complexity by dividing the scheduling problem into subsections based on its process recipes. Since we only have to deal with the resources, tasks, and equipment associated with each subsection, the individual subproblem sizes decrease significantly.

Introduction of intermediate resource purchase variables, in a manner similar to the heirarchical constraint modification approach presented earlier, allows complex processing chains to be split into more manageable sections. As an example, refer to Figure 12 and apply this approach. Assume that all the resources shown in the figure are storable. The two intermediates (I1 and I2) are the only storable resources along the critical path, in addition to the final products, so they are the obvious choice for resource decomposition giving four distinct sections. The section producing P2 is independent of the three sections needed to produce P1, so we will arbitrarily solve this section first. Solution of this section based on a demand for P2 of 4,000 lb (1.8 Mg), due at the end of the horizon (time interval 180), leads to the schedule given in Figure 14. This schedule restricts production within future sections to times not already in use for P2 production. Next, purchase variables are introduced for I2 (treating it as a raw material) and the scheduling problem associated with the third section is determined based on a demand for P1 of 3,750 lb (1,700 kg), due at the end of the horizon. The solution to this problem gives us two results: a "purchase" schedule for I2 and a production schedule for both the third and fourth sections. This "purchase" schedule for I2 in turn becomes the "demand" schedule for the second section. This new problem is solved with purchase variables for I1, giving the "purchase/demand" schedule for I1 and production schedule for sections 2, 3 and 4. This is continued for the final section, giving us the schedule for the full problem (Figure 15).

The time needed to obtain a feasible solution using resource decomposition is compared in Table 11 to the time needed if the problem is solved all at once. For this example, the overall solution time using resource decomposition is significantly lower than the time needed to solve the global

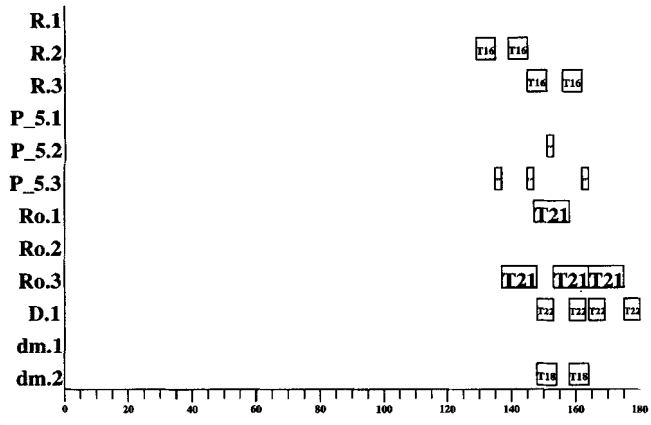

Figure 14. Schedule for P2 production.

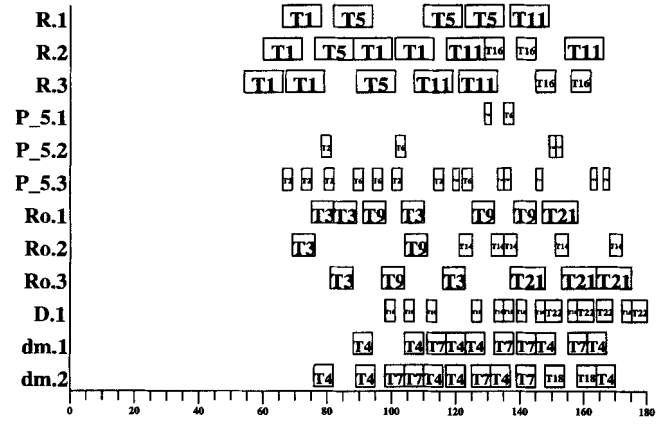

Figure 15. Overall schedule for P1 and P2 production.

Table 11. Timing Results for Resource Decomposition Example

Section	Integer Variables	Solution Time* (cpu s)
All-at-once	**7,855**	**646.89**
4	2,098	23.31
3	2,266	28.08
2	1,958	20.17
1	1,318	12.40
Total	**7,640**	**83.96**

*Solved to first feasible solution on HP 9000/735.

problem. This is due to the reduction in the number of tasks, units, and resources that must be considered within each subproblem.

Resource decomposition can, however, introduce a number of difficulties that were not encountered in the example presented. If a section is overoptimistic concerning the availability of its "raw material," there needs to be a feedback mechanism to incorporate this information into future solutions. Also, for problems with widely spaced demands, it is important to ensure that a campaign assumption is not inadvertently introduced. This can lead to capacity underestimation and problem infeasibility.

There is no reason why resource decomposition cannot be introduced into the rolling window approach. In some ways, resource decomposition mimics the behavior of the disaggregation heuristic already used in conjunction with the rolling window. Both reduce the number of tasks and units dealt with at any one time, but resource decomposition reduces the resources considered as well. It is also possible to utilize both methods together to further reduce the problem size. An example would be a problem with multiple sections and, within these sections, multiple production pathways. Resource decomposition could be used to select a section to schedule and then the disaggregation heuristic could be used to only generate variables associated with the most viable production pathway within the section.

Task-unit aggregation

A possible approach that attempts to tackle both task/resource and unit aggregation at once is the use of supertasks and superunits. Supertasks and superunits are intimately related and were alluded to in Bassett et al. (1996) and Zentner et al. (1994). In these articles unit aggregation was performed by assuming all the units together as a single superunit. For each product a supertask was created that consisted of all the tasks needed to make the product. To determine the batch sizes and processing times of the supertask on the superunit, it was assumed that the plant only made one product (ran a single supertask) for a prespecified amount of time. In doing this, strong assumptions were made concerning the running of the equipment, which may or may not always be valid.

It may be possible to utilize supertasks without needing to generate superunits. This would remove the need for having to assume only one product being produced at a time. Instead, a supertask would specify what equipment is being utilized at what point during its execution and its maximum batch size calculated based on this information. Even for a small example this can lead to a large number of possible supertasks. Looking at the small example given in Figure 16, it is easy to see that a number of supertasks can be generated for a given recipe. Even though six supertasks are generated, this is a third less than the original number of assignment variables. For problems with long processing chains, the use of supertasks may significantly reduce the number of assignment variables generated.

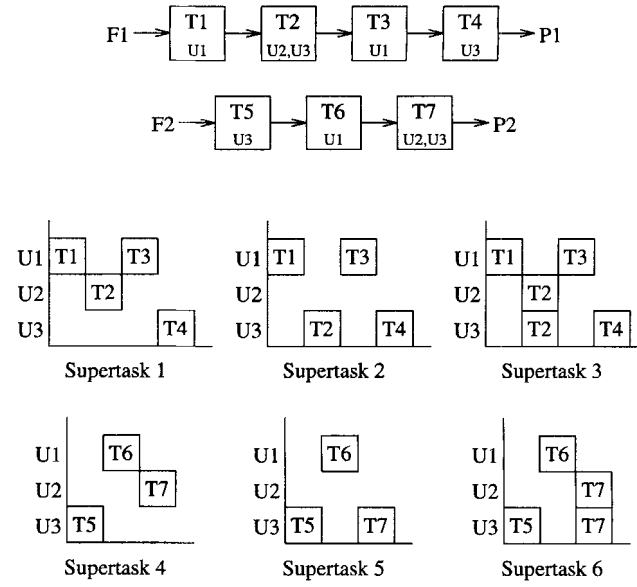

Figure 16. Supertasks for a small example.

Conclusions

In this article we have presented a number of decomposition methods for the solution of large-scale scheduling problems. All of the methods attempt to reduce the problem complexity through decomposition of time, units, tasks, or resources. Using some of these methods we have solved problems of a larger scale than have been attempted previously in the literature. In Table 12 the strengths and weaknesses of the four approaches are presented in binary form. An explanation of the situations represented in the table should help to better differentiate the approaches.

First we clarify the terminology used. A *production chain* is the processing time between storable resources. This could be from raw materials to an intermediate or all the way to final product. A *production run* is the minimum amount of time that is necessary to meet a future demand. So the larger the quantity of material requested, the longer the production run needed. *Equipment utilization* is a measure of how busy a facility is. The busier the facility, the harder it is to schedule. *Demand spacing* and *number of products* are self-explanatory.

All of the approaches work reasonably well in scheduling frequently occurring, low-level demands for a few products that have short processing stages and are produced using an underutilized facility. As the number of products increase, all of the algorithms have a hard time keeping up with the increased complexity due to additional tasks, units, and/or resources. The usefulness of resource decomposition will be in reducing the number of products being scheduled at any given time to a more manageable number.

Table 12. When to Use (√) and Not Use (X) Various Decomposition Approaches

Method ⇒ Situation ⇓	Constraint Modification	Constraint Disaggregation	Rolling Window	
			Forward	Reverse
Short production chains	√	√	√	√
Long production chains	X	X	√	√
Short production runs	√	√	√	√
Long production runs	X	X	X	√
Light equipment utilization	√	√	√	√
Heavy equipment utilization	X	X	√	√
Closely spaced demands	√	√	√	√
Widely spaced demands	√	√	X	√
Few products	√	√	√	√
Many products	X	X	X	X

For longer production chains, constraint modification and disaggregation approaches begin to falter. This is because these approaches require the aggregate intervals to be at least as long as the longest production chain. This leads to large scheduling subproblems that must be solved, possibly, numerous times before a feasible solution is found. The rolling-window approaches, on the other hand, only require that the window be as long as the longest *task* processing time; consequently, longer production chains have little effect.

The artificial end effects introduced by constraint modification and disaggregation lead to difficulties with problems having long production runs and heavy equipment utilization. With the constraint-modification approach, numerous iterations are needed and the solution obtained will have equipment locally idle while a global solution would require production. The constraint disaggregation approach fails because the hybrid planning/scheduling problem increases substantially as infeasible subproblems are added.

The forward-rolling window fails for long production runs and widely spaced demands due to the large window size needed in order to anticipate the demands early enough to meet them. The reverse rolling-window is able to overcome these shortcomings by beginning to schedule backwards from the latest demand, that is, implicitly anticipating demands.

Thus, the most successful approach presented is the reverse-rolling window coupled with a disaggregation heuristic. Together these methods significantly reduce the time, task, and unit complexity that have to be dealt with at one time. Further study of other decomposition/aggregation techniques, such as resource decomposition or task-unit aggregation, will surely help to further reduce problem complexity and solution time.

Notation

Sets

Group = set of all units that are identical
I = set of all tasks
I_j = set of all tasks that can run on unit j
J = set of all units
J_i = set of all units that can run task i
J_s = set of all units that can store resource s
S = set of all resources
$S^{\text{intermediate}}$ = set of all intermediate resources
T = set of all aggregate time periods
T' = set of all discrete time periods

Parameters

C_{ijt}^{fixed} = fixed cost to run task i on unit j at time t
$C_{ijt}^{\text{variable}}$ = variable cost to run task i on unit j at time t
C_{sjt}^{store} = cost to store resource s in unit j at time t
C_{st}^{buy} = cost to buy resource s at time t
D_{st} = demand for resource s during aggregate interval t
$d_{st'}$ = demand for resource s at discrete time t'
f_{ijs}^{consume} = fraction of task i on unit j that consumes resource s
f_{ijs}^{create} = fraction of task i on unit j that creates resource s
H_t = length of aggregate interval t
$p_{ijt'}$ = processing time for task i on unit j at discrete time t'
$V_{ij}^{\max}$ = maximum production quantity for task i on unit j
$V_{ij}^{\min}$ = minimum production quantity for task i on unit j

Variables

A_{sjt} = quantity of resource s stored in unit j at aggregate time t
$a_{sjt'}$ = quantity of resource s stored in unit j at discrete time t'
B_{ijt} = production quantity for task i on unit j at aggregate time t
$b_{ijt'}$ = production quantity for task i on unit j at discrete time t'
$e_{ijt'}$ = excess production quantity for task i on unit j at discrete time t'
E_{ijt} = excess production quantity for task i on unit j at aggregate time t
QB_{st} = quantity of resource s purchased at aggregate time t
$qb_{st'}$ = quantity of resource s purchased at discrete time t'
W_{ijt} = assignment variable for task i on unit j at aggregate time t
$w_{ijt'}$ = assignment variable for task i on unit j at discrete time t'

Literature Cited

Bassett, M. H., F. J. Doyle III, G. K. Kudva, J. F. Pekny, G. V. Reklaitis, S. Subrahmanyam, M. G. Zentner, and D. L. Miller, "Perspectives on Model-Based Integration of Process Operations," *Proc. Comput. Chem. Eng.*, **20**, 821 (1996).

Bitran, G. R., E. A. Haas, and A. C. Hax, "Hierarchical Production Planning: A Single Stage System," *Oper. Res.*, **29**, 717 (1981).

Bitran, G. R., and A. C. Hax, "On the Design of Hierarchical Production Planning Systems," *Decis. Sci.*, **8**, 28 (1977).

Bowman, E. H., "The Scheduling Sequencing Problem," *Oper. Res.*, **7**, 621 (1959).

Kondili, E., C. C. Pantelides, and R. W. H. Sargent, "A General Algorithm for Scheduling Batch Operations," *Proc. Int. Symp. on Process Systems Eng.*, Sydney, Australia, p. 62 (1988).

Kondili, E., C. C. Pantelides, and R. W. H. Sargent, "A General Algorithm for Short-Term Scheduling of Batch Operations. Part I —Mathematical Formulation," *Comput. Chem. Eng.*, **17**, 211 (1993).

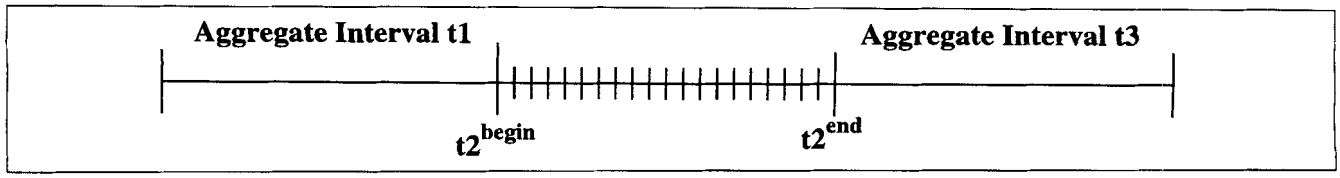

Figure A1. Example with no crossover variables.

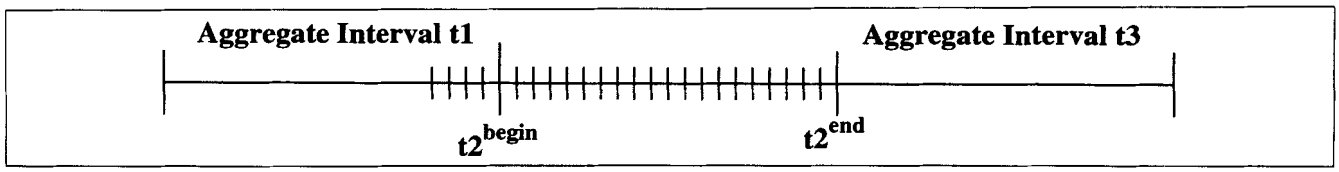

Figure A2. Example with crossover variables.

Ku, H., D. Rajagopalan, and I. Karimi, "Scheduling in Batch Processes," *Chem. Eng. Prog.*, 35 (1987).

Manne, A. S., "On the Job-Shop Scheduling Problem," *Oper. Res.*, **8**, 219 (1960).

Miller, D. L., H. Singh, and K. A. Rogers, "A Modular System for Scheduling Chemical Plant Production," *Proc. Int. Conf. on the Foundations of Computer Aided Process Operations, FOCAPO II*, p. 355 (1993).

Musier, R. F. H., and L. B. Evans, "Batch Process Management," *Chem. Eng. Prog.*, 66 (1990).

Pantelides, C. C., "Unified Framework for Optimal Process Planning and Scheduling," *Proc. Conf. of Foundations of Computer Aided Operations*, p. 253 (1994).

Pekny, J. F., and M. G. Zentner, "Learning to Solve Process Scheduling Problems: The Role of Rigorous Knowledge Acquisition Frameworks," *Proc. Int. Conf. on the Foundations of Computer Aided Process Operations*, p. 275 (1993).

Sahinidis, N. V., and I. Grossmann, "Reformulation of Multiperiod MILP Models for Planning and Scheduling of Chemical Processes," *Comput. Chem. Eng.*, **15**, 255 (1991).

Shah, N., C. C. Pantelides, and R. W. H. Sargent, "A General Algorithm for Short-Term Scheduling of Batch Operations: II. Computational Issues," *Comput. Chem. Eng.*, **17**, 229 (1993).

Subrahmanyam, S., J. F. Pekny, and G. V. Reklaitis, "Design of Batch Chemical Plants under Market Uncertainty," *Int. Eng. Chem.*, **33**, 2688 (1994).

Subrahmanyam, S., G. K. Kudva, M. H. Bassett, and J. F. Pekny, "Applications of Distributed Computing to Batch Plant Design and Scheduling," *AIChE J.*, **42**, 1648 (1996a).

Subrahmanyam, S., J. F. Pekny, and G. V. Reklaitis, "Decomposition Approaches to Batch Plant Design and Planning," *Ind. Eng. Chem. Res.*, accepted (1996b).

Wilkinson, S. J., N. Shah, and C. C. Pantelides, "Scheduling of Multisite Flexible Production Systems," AIChE Meeting, San Francisco (1994).

Zentner, M. G., J. F. Pekny, G. V. Reklaitis, and J. N. D. Gupta, "Practical Considerations in Using Model-based Optimization for the Scheduling and Planning of Batch/Semicontinuous Processes," *J. Process Contr.*, **4**, 259 (1994).

Appendix: Formulation with and without Crossover Variables

To show the differences between the formulations with and without crossover variables, material balance constraints are presented for each case. Figure A1 shows the situation when crossover variables are *not* introduced, while Figure A2 shows the situation when they are introduced.

When crossover variables are not introduced, $b_{ijt'}$ are only generated for $t2^{\text{begin}} + p_{ijt'} \le t' < t2^{\text{end}} - p_{ijt'}$, within region $t2$. This leads to this modified UDM material balance constraint:

$$\sum_{j \in J_s} a_{sjt'} - \sum_{i \in I} \sum_{j \in J_i} b_{ij(t'|t'+p_{ijt'} < t2^{\text{end}})} f_{ijs}^{\text{consume}} - d_{st'} = \sum_{j \in J_s} a_{sjt'-1} + qb_{st'} + \sum_{i \in I} \sum_{j \in J_i} b_{ij(t'-p_{ijt'} \ge t2^{\text{begin}})} f_{ijs}^{\text{create}} \quad \forall s \in S \quad \forall t' \in t2. \tag{A1}$$

Since no tasks for region $t2$ interact with $t1$ or $t3$, the material balance constraints for the aggregate intervals are unchanged from those given by Eq. 7.

For the case when crossover variables are introduced, $b_{ijt'}$ are generated for $t2^{\text{begin}} - p_{ijt'} \le t' < t2^{\text{end}}$. Based on variables generated, the UDM material balance constraints are unchanged from the ones presented in the subsection titled "Uniform Discretization Model Formulation." The aggregate intervals, however, must be modified to take into account tasks that either begin or end within their domain. So for aggregate interval $t1$, the modified material balance is given as

$$\sum_{j \in J_s} A_{sjt1} + \sum_{i \in I} \sum_{j \in J_i} (B_{ijt1} + b_{ij(t'-p_{ijt'} < t1^{\text{end}})}) f_{ijs}^{\text{consume}} + D_{st1} = \sum_{j \in J_s} A_{sjt1-1} + QB_{st1} + \sum_{i \in I} \sum_{j \in J_i} B_{ijt1-p_{ijt1}} f_{ijs}^{\text{create}} \quad \forall s \in S \tag{A2}$$

and for $t3$, is given as

$$\sum_{j \in J_s} A_{sjt3} + \sum_{i \in I} \sum_{j \in J_i} B_{ijt3} f_{ijs}^{\text{consume}} + D_{st3} = \sum_{j \in J_s} A_{sjt3-1} + QB_{st3} + \sum_{i \in I} \sum_{j \in J_i} \times (B_{ijt3-p_{ijt3}} + b_{ij(t'+p_{ijt'} > t3^{\text{begin}})}) f_{ijs}^{\text{create}} \quad \forall s \in S. \tag{A3}$$

Manuscript received Feb. 1, 1996, and revision received May 21, 1996.