Combinatorial Framework for Effective Scheduling of Multipurpose Batch Plants

E. Sanmartí and L. Puigjaner
Chemical Engineering Dept., Universitat Politècnica de Catalunya, E.T.S.E.I.B., Diagonal 647, E-08028 Barcelona, Spain

T. Holczinger and F. Friedler
Dept. of Computer Science, University of Veszprém, Veszprém, Egyetem u. 10, H-8200, Hungary

The production scheduling of multipurpose batch plants is considered. A novel graph representation that looks at the specific characteristics of production scheduling in chemical processes is proposed. In these graphs, the nodes represent the production tasks, the arcs of the precedence relationships among them. The representation is flexible enough to consider a wide variety of production structures, including complex recipes. Both nonintermediate storage (NIS) and unlimited intermediate storage (UIS) transfer policies can be considered simply by choosing the appropriate precedence relationships. This representation provides the opportunity of incorporating highly efficient graph algorithms together with an appropriate branch-and-bound algorithm for solving multipurpose scheduling problems. The efficiency of the proposed method is established by solving examples and a complex case study.

Introduction

Multipurpose batch plants are characterized by their flexibility and ability to produce a large number of different products in different qualities so as to satisfy the client's demand, and by the possibility of considering alternate production paths for the same product in order to reduce costs. This high degree of flexibility is one of the major sources of the complexity of scheduling problems. In practice, a large number of variables have to be introduced (e.g., for tasks to equipment assignments, products, and/or task sequencing and task timing) to describe the problem appropriately in a mathematical programming model that is difficult or frequently impossible to solve by available general-purpose solvers. Simplifying assumptions such as reducing the search space to permutation schedules (see, for example, Reklaitis, 1981), in which complete batches are sequenced instead of tasks, may result in valuable solutions; nevertheless, optimality cannot be guaranteed in most cases.

Short-term scheduling of chemical multipurpose batch plants has similarities with the general job-shop problem, which has been widely treated in operations research. The approach traditionally used is based on a graph representation of the scheduling problem combined with a branch-and-bound (B&B) algorithm (see, e.g., Adams et al., 1988; Carlier and Pinson, 1989). For the job-shop scheduling problem, B&B algorithms are usually coupled with specific heuristics that greatly accelerate the convergence with the optimal or near-optimal solutions. The scheduling of chemical batch plants, however, substantially differs from the job-shop scheduling problems. The former is usually more complex than the latter, since additional conditions have to be considered, for example, the storage of intermediate liquid materials that do not occur in the discrete mechanical manufacturing industry. Another key difference is the presence of unstable intermediate products that may require immediate transfer. These differences prevent the direct adoption of the graph representation and algorithms developed for job-shop scheduling to batch chemical processes. Thus, batch chemical systems are usually scheduled by using other techniques, mainly mathematical programming (see, for example, Voudouris and Grossmann (1994) and Sanmartí et al. (1996)), sequencing and scheduling via tailored heuristics or stochastic (simulated annealing, genetic algorithms) methods (see, for example, Kudva et al. (1994), Graells et al. (1996), Hasebe et al. (1996), and Murakami et al. (1997)).
The selection of the representation technique in solving any complex problem like scheduling has major importance for several reasons [see, for example, state-task network (STN) representation by Kondili et al., 1993; resource-task network (RTN) representation by Schilling and Pantelides, 1996; event operation network (EON) representation by Graells et al., 1998]. A convenient representation may provide the opportunity of exploiting the unique features of the problem, resulting in a procedure with a higher efficiency. Furthermore, if the representation explicitly expresses the crucial points, it may contribute to an understanding of the depth of the problem that may facilitate the development of new algorithms or the improvement of available ones. The generality of the chosen representation technique also determines the range of problems that can be treated.

Kondili et al. used a discrete representation of time and introduced the state-task network representation of the process. The major advantage here is the ability to consider very general-process recipes involving batch splitting and mixing, material recycles, storage policies, resource constraints, and so on. The formulation of Kondili et al. is based on the definition of binary variables that indicate whether tasks start in specific pieces of equipment at the start of each time period. However, the major problems with this approach are the large size of the resultant mixed-integer linear programming (MILP) model and mapping the discretization points with the actual points in time when the events take place. This forms the basis of several other pieces of research that are aimed at taking advantage of the representational capabilities of the formulation while improving its numerical performance. Sahinidis et al. (1991) disaggregated the model in a way that solution efficiency is improved despite the large nature of this model. Shah et al. (1993) modified the allocation constraints to generate the smallest possible integrality gap for this type of formulation. In addition, Elkamel (1993) proposed a heuristic decomposition method to solve separate scheduling problems for parts of the overall scheduling problem. In a later work, Yee and Shah (1997, 1998) considered the further use of heuristics embedded in the formulation. Pantelides et al. (1994) presented a critique of the STN and associated scheduling formulations. They then proposed an alternative representation, the resource-task network (RTN), based on a uniform description of all resources. Here tasks are assumed to only consume and produce resources instead of materials.

As for continuous representation of time, Zhang and Sargent (1994, 1996), presented a continuous-time formulation based on the RTN representation. They composed an MILNP representation that was solved using the liberalization procedure. Schilling and Pantelides (1996) proposed a hybrid B&B solution procedure that works well in reduced scenarios. Pinto and Grossmann (1995) used preordering constraints and a decomposition scheme for large systems that minimize earliness and eliminates unnecessary setups. Graells et al. (1998) presented the EON representation that simplifies the timing subproblem. Together with the process-material network (PMN) representation, the EON provides a realistic and robust framework for detailed process simulation at the scheduling level.

A large portion of the most recent research in scheduling relates to the development of mathematical models as the best way of representing the complex interactions between resource allocation, task timing, materials flows, and equipment capacities (see, for example, recent reviews in this area by Shah, 1998; Pekny and Reklaitis, 1998; Pinto and Grossmann, 1998; Puigjaner, 1999). In the present work, a new graph representation appropriate for combinatorial algorithms is introduced. The article describes a general framework for solving different types of multipurpose batch scheduling problems. Therefore, it provides a good basis for effective algorithms to solve a large variety of scheduling problems.

Problem Definition

Three types of information define a multipurpose batch-scheduling problem; these are the recipes of each product, tasks to equipment assignments, and the amount to be produced from each product.

A recipe is an entity that contains the minimum set of information that uniquely defines the manufacturing requirements for a specific product. The ISA SP88 standard defines four levels for the types of recipes that are used in batch processes, depending on the user requirements. These levels are:

- General recipe identifies raw materials, their relative quantities, and the required processing. It is considered as enterprise-level information, and is therefore equipment independent. Besides it is not specific to a particular site.
- Site recipe is a combination of site-specific information and the general recipe (it is specific to a particular site).
- Master recipe includes information about required process cell equipment, raw materials with corresponding quantities, and the procedure for making the product. It can be derived from a general recipe or a site recipe.
- Control recipe includes additional information about the process units that are used in manufacturing a single batch of a single product. This recipe is created from the master recipe when a batch is scheduled for production.

Based on the four types of recipes, the master recipe is to be considered for the scheduling of batch processes. This master recipe can be conveniently represented as a directed graph, where the nodes represent the production tasks and the arcs the precedence relationships among them. Note that the production time (PT) and the set of plausible equipment units (Eq.) of a task are given at the corresponding node. Figure 1 illustrates the conventional representations of the recipe of a product composed by three reaction stages. Natur-
rally, complex recipe structures can also be represented in this way, as illustrated in Figure 2, where, for example, two intermediates are produced, mixed, and further processed in the production of A.

This conventional representation is similar to the flow-sheet representation of continuous plants (see Reklaitis (1991)), the only role of the arcs being to express the order of tasks; all other information is related to the nodes of the graph. For combinatorial algorithms of scheduling, however, it is beneficial to distribute information between nodes and arcs; processing times are assigned to the arcs instead of nodes. Figure 3 shows the graph-representation of recipes given on Figure 2, where $S_i$ denotes the set of those equipment units that can perform the task represented by node $i$ (such as $S_1 = \{1,2\}$); in addition, an additional node (the so-called product node) is introduced for each product. In this representation, the value assigned to an arc expresses the lower bound for the difference between the starting times of the two related tasks. The processing time of a task may be different for different equipment units. In this case, the weight of the arc is the minimum of the processing times of the plausible equipment units.

Formally, any recipe can be represented by an acyclic directed graph. Note that recycling in the process flow sheet does not result in a cycle in the graph of the recipe, since it is realized as a feedforward to the production of the next batch or some batch to be started later.

**Graph Representations for Solving Scheduling Problems**

Scheduling of batch chemical plants has been traditionally approached by tailored heuristics and mathematical programming, either MILP or mixed-integer nonlinear programming (MINLP). The graph-theoretic approach has also been used frequently in solving complex combinatorial problems, including scheduling. These applications in scheduling, however, have been limited to the general job-shop-scheduling problem of the mechanical industry where intermediates can be stored between operations, that is, for an unlimited intermediate storage (UIS) policy. In solving a chemical scheduling problem, however, the liquid, and sometimes the unstable nature of the intermediate products, has to be considered. Unlike solids, liquid intermediates have to be stored in specific vessels that have to be available at each specific equipment unit. Consequently, more sophisticated graph representation and algorithms are required in order to represent this situation. Furthermore, liquid intermediates are frequently unstable, and they have to be transferred immediately after processing.

**Graph representation for the unlimited intermediate storage policy**

Assume that the number of batches of each product to be produced is already known and definite assignments of equipment to tasks are given. Then, the sequence of the tasks to be processed can be represented in a graph (Adams et al., 1988) where the task sequence of all equipment units is defined by a set of conjunctive arcs that connect the tasks assigned to the same equipment unit. These arcs have the same weight as the processing time of the tasks that introduce a set of precedence constraints in addition to the constraints imposed by the recipe. For example, 1–6–7 is the task sequence of equipment unit E1 shown in Figure 4b for the scheduling problem specified by the recipe on Figure 4a. In this example, E1 is assigned to each of tasks 1, 6, and 7 where, $E_1 \in S_1$, $S_6$, $S_7$. It can be observed in Figure 4b that task 6 is subject to two precedence constraints: it cannot start before task 5 is completed (because of the recipe) and before task 1 has finished (because of the sequence).

This type of graph representation is useful in solving the job-shop problems with the UIS policy; however, it may not be appropriate for nonintermediate storage (NIS) cases. For the former, it is assumed that an equipment unit is available immediately after it processes a task, that is, the intermediate product that has been generated in this task is removed from the equipment unit and stored until the next task in the recipe starts its processing. In most of the chemical batch processes, however, the NIS transfer policy is to be followed instead of the UIS policy.

**S-graph for nonintermediate storage policy**

In the NIS case, an equipment unit is not free after processing a task until the material stored in it has been transferred to the equipment unit assigned to the next task in the recipe. Arc or arcs express these additional constraints imposed by the NIS policy. Let $\tau_j$ denote the set of tasks that follow task $j$ according to the recipe. If equipment unit $E_i$ is assigned to task $j$ after completion of task $k$, then a zero-
weighted arc (or an arc whose weight is equal to the length of the changeover time if applicable) is established from each element of \( \tau_j \) to \( k \). This method of representation is called S-graph. The task sequence of equipment E1 given in Figure 4b is given via S-graph in Figure 5. Instead of connecting the node of tasks 1 to 6, and then that of 6 to 7 by arcs weighted by the length of the processing time, as shown in Figure 4b, zero weighted arcs are used to connect the node of tasks 2 to 6 and that of tasks 11 to 7, respectively.

A feasible schedule for the UIS transfer policy may not be feasible for the NIS case. This infeasibility can be detected in the S-graph representation by finding a directed cycle in the graph, while it cannot be recognized on the conventional representation. For example, Figure 6 shows the Gantt chart of an infeasible schedule of the scheduling problem specified by the recipe in Figure 4a for the NIS policy; nevertheless, it is an optimal schedule for the UIS policy. The Gantt chart given in Figure 6 shows the transfer of material from equipment unit E1 (task 1) to equipment unit E3 (task 2) simultaneously with the transfer from E3 (task 5) to E1 (task 6). This transfer of material can only be performed if an intermediate storage is available to store one of the products while the other is being transferred, which is the case of the UIS policy. An optimal solution for the NIS policy of the same example is given in Figure 7.

Although conventional representation is not convenient for NIS, an S-graph can represent both NIS and UIS scheduling problems correctly. For simplicity, in this article the mathematical formulation and the algorithms will be introduced for only the NIS case.

**Mathematical Formulation of the S-Graph**

Formally, directed graph \( G \) can be given as a pair \( (N, A) \), where \( N \) is a finite set, the set of nodes, and \( A \) is a set of pairs of nodes identifying the arcs of the graph (that is, \( A \subseteq N \times N \)). In an S-graph, two classes of arcs, the so-called recipe-arcs and schedule-arcs, are specified. Therefore, an S-graph is given in the form of \( (G(N, A_1, A_2), \) where \( N, A_1, \) and \( A_2 \) denote the sets of nodes, recipe-arcs, and schedule-arcs, respectively. It is supposed that \( A_1 \subseteq N \times N, A_2 \subseteq N \times N, \) and \( A_1 \cap A_2 = \emptyset \); furthermore, a nonnegative value, \( c(i,j) \), which denotes the weight of arc \((i,j)\), is assigned to each arc. In practice, if the arc is established from node \( i \) to node \( j \), that is, \((i,j) \in A_1 \cup A_2\), then it is supposed that the task corresponding to node \( j \) cannot start its activity earlier than \( c(i,j) \) after the task corresponding to node \( i \) starts. Specific types of S-graphs are identified for a recipe (that is, recipe-graph) and for a schedule of all tasks (that is, schedule-graph).

**Recipe-graph**

A recipe defines the order and type of tasks, the material transfers among them, and the set of plausible equipment units of each task. This type of information should be represented by the graph of a recipe.
Let one node be assigned to each task (task node) and one to each product (product node). An arc is established between the nodes of consecutive tasks determined by the recipe, and one is also established from each task node that generates the product to the corresponding product node. The weight of an arc is specified by the processing time of the task corresponding to the initial node of the arc if a single equipment unit is available for this task and the minimum of the processing times of all equipment unit if more than one equipment unit is available.

If multiple batches of a product are to be produced, the task nodes, the product nodes, and the arcs are multiplied appropriately. The resultant graph is called task network, where \( N_t \) and \( N_p \) denote the set of its task nodes and product nodes, respectively.

In practice, the inputs of a task may not need to be available at the same time; instead, further constraints may need to be applied on the timing of these inputs. Stated formally, the timing of the feeds is to be given as a partial order on their starting times. In the simplest case, all inputs must be available simultaneously. In Figure 8, according to this representation technique, task 4 has three different inputs that are produced in parallel by tasks 1, 2, and 3. Obviously, three pieces of equipment must be prepared simultaneously to feed task 4.

In general, the practical order of the inputs of a task is given by the so-called feed-precedence graph of the task. For example, suppose that input 1 precedes input 3, and input 1 is available simultaneously with input 2 in the part of a task network given in Figure 9a. Additional nodes and arcs express the order of precedence of the feeds of task 4 according to Figure 9b.

The task network extended with a feed-precedence graph is called recipe-graph. It represents all structural information given in a recipe. If \( G(N_t, A_t, A_s) \) is a recipe-graph, it is acyclic, \( N_t = N_{t_1} \cup N_{t_2} \), and \( A_s = \emptyset \).

For simplicity, the partial ordering of the timing of the feeds of a task is not examined here (for more details of the feed-precedence graph, see Appendix A). Let \( N_{t_i} \cap N_{p_i} = \emptyset \). It is supposed that there is at least one equipment unit available that can be assigned to a task, that is, the set of task nodes can be given as \( N_t = N_{t_1} \cup N_{t_2} \cup \cdots \cup N_{t_n} \), where \( N_{t_i} \) and \( N_{p_i} \) (\( i, j = 1, 2, \ldots, n \)) are not necessarily disjoint.

**Example 1.** Suppose that two batches of product A and one batch of product B are to be produced, where product A is produced in two consecutive steps. The first step can be performed by any equipment unit given in set \( S_1 \), and step 2 can be performed by any equipment unit given in set \( S_2 \). Product B is produced in three consecutive steps that can be performed by any of the elements of sets \( S_3 \), \( S_4 \), and \( S_5 \), respectively. The resulting recipe-graph is given in Figure 10.

**Schedule-Graph**

A specific S-graph, termed schedule-graph, is introduced to describe a single solution of a scheduling problem; there...
exists one schedule-graph for each feasible schedule of the problem. S-graph G′(N, A1, A2) is called a schedule-graph of recipe-graph G(N, A1, ∅), if all tasks represented in the recipe-graph have been scheduled by taking equipment-task assignments into account. The schedule-graph of the optimal schedule can be effectively generated by an appropriate search strategy, as will be shown later.

Example 2. Three products are to be produced according to the recipe given in Figure 11. One equipment unit can be assigned to each task according to the following: S1 = (E1), S2 = (E3), S3 = (E2), S4 = (E2), S5 = (E3), S6 = (E1), S7 = (E1), S8 = (E2), and S9 = (E3). There are eight different solutions represented by the eight schedule-graphs given in Figure 12.

For the formal definition of a schedule-graph, suppose that S-graph G′(N, A1, A2) is given where G(N, A1, ∅) is a recipe-graph and A2 ⊆ N × N. Let Mj (i = 1, 2, ..., n) denote the set of nodes of those tasks that are to be performed by equipment unit i according to S-graph G′(N, A1, A2). It is assumed that exactly one equipment unit is assigned to each task, that is, Mj ∩ Ml = ∅ (i ≠ j, i, j = 1, 2, ..., n). Therefore, M1, M2, ..., Mn is such a partitioning of the set of task nodes Mj = Nj ∪ N2 ∪ ... ∪ Nn, that Mj ⊆ Nj (i = 1, 2, ..., n).

A specific graph, called component-graph, will be identified to show the activity of each equipment unit. Stated formally, S-graph G′(N′, A1i, A2i) (⊆ G′(N, A1, A2)) is the component-graph of equipment unit i (i = 1, 2, ..., n) if

- N′ includes all nodes of Mj and the end nodes of all arcs in A1 starting from any elements of Mj (i.e., N′ = Mj ∪ {k : (j,k) ∈ A1, j ∈ Mj});
- A1i includes all recipe-arcs of G′, starting from any elements of Mj (i.e., A1i = {(j,k) : (j,k) ∈ A1, j ∈ Mj});
- A2i includes all schedule-arcs of G′, starting from the end node of an arc of A1i and pointing to any elements of Mj (i.e., A2i = {(j,k) : (j,k) ∈ A2, k ∈ Mj, and ∃l Ml such that (i,l) ∈ Ai});
- S-graph G′(N, A1, A2) is defined to be a schedule-graph for recipe-graph G(N, A1, ∅) and equipment assignments Mi (i = 1, 2, ..., n) if it satisfies the following four axioms.

(SG1) G′ is acyclic
(SG2) G′ defines total order on Mi ∪ {j} for all j ∈ N′ and i = 1, 2, ..., n.
(SG3) The out-degree of every node of N′ (i = 1, 2, ..., n) is at most 1 according to the arcs in A2i.
(SG4) Schedule-graph G′ is the union of its component schedule-graphs, that is, G′ = ⋃i=1 G′i.

It is important to examine the relation between a schedule-graph and the set of feasible schedules of a scheduling problem. Solving a practical problem, arc (i,j) of an S-
Figure 13. Component-graphs of schedule-graph No. 2 of Figure 12.

Figure 14. Main procedure of the scheduling algorithm.

Figure 15. Branching procedure of the scheduling algorithm.
The bounding procedure given in Figure 16 first tests the feasibility of a partial problem. If this test is proved positive, it determines a lower bound for the makespan of all solutions that can be derived from this partial problem. If no waiting time is limited, a schedule-graph is feasible if and only if it is acyclic. Thus, the feasibility of a partial problem is tested by a cycle search algorithm. Note that it can be realized by an efficient (polynomial) algorithm (see Appendix B for the algorithm). If the waiting time is limited for some intermediate products, further examination is required, for example, by solving an LP problem.

Obviously, the S-graph of a partial problem is a subgraph of the S-graph of any of its child partial problems. Since an additional arc does not decrease the length of the longest path of a graph, the length of the longest path of any subgraph of a schedule-graph is a lower bound of the longest path of the schedule-graph. Moreover, the length of the longest path of a schedule-graph is also a lower bound of the makespan of the corresponding solution. Consequently, the following proposition is true.

Proposition 1. The longest path of any subgraph of a schedule-graph provides a lower bound of the makespan of the solution related to the schedule-graph.

The longest path of a graph can be generated efficiently by a polynomial time algorithm; thus, it can be applied effectively in solving scheduling problems (see Appendix B for the algorithm).

At certain partial problems of the B&B algorithm, especially at the highest levels of the enumeration tree, the longest path may not give a sharp lower bound simply because no or only few schedule arcs are considered. In general, this bound can be sharpened by solving a specific LP problem based on the result of the longest-path algorithm (see Appendix C for the details) in return of an additional computational effort.

If the waiting times of some intermediate products are limited, LP relaxation is to be used to test the feasibility and generate a lower bound. Each time that the lower bound of the partial schedule of a node is greater than the current upper bound or the partial schedule is cyclic (i.e., it is structurally infeasible), this partial schedule and the corresponding node on the enumeration tree are pruned.

Illustrative example

The recipe-graph is shown in Figure 17, where one batch of each of the two products is to be generated. The search tree is depicted in Figure 18. The nodes of the tree represent the partial problems, and they are numbered in the order in which the proposed base algorithm generates them. The branches are identified by the task sequence they introduce. The lower bound associated with each of the feasible partial problems is shown in boldface. The node of an infeasible partial problem is crossed, such as No. 4 and No. 7. The S-graphs that correspond to these partial problems are shown in Figure 19.

The root of the search tree (partial problem No. 1) corresponds to the recipe-graph, the longest path, that is, the lower bound of its makespan is 9. From this partial problem, two branches, partial problems No. 2 and No. 3, are generated for the sequence of equipment unit E1: 1–3 and 3–1, with a lower bound of 12 and 11, respectively. The branch with the best lower bound is partial problem No. 3, and it is selected to continue expanding the tree. From this partial problem, two branches (partial problems No. 4 and No. 5) are generated: sequences 2–4 and 4–2 for equipment unit E2. Both partial problems are complete schedules, that is, they can lead to a new upper bound for the minimum makespan. Partial problem No. 4 contains a cycle; it must be discarded. Partial problem No. 5 represents a complete schedule with a makespan of 13, that is, it is an upper bound for the optimal makespan. Partial problem No. 2 is the next branch to be expanded, from which sequences 2–4 and 4–2 for equipment unit E2 are generated (partial problems No. 6 and No. 7); both are complete schedules. Partial problem No. 6 corre-
sponds to a schedule with a makespan of 12; it is better than the current upper bound, and accordingly is set as the new upper bound of the minimum makespan. Partial problem No. 7 is not explored, since the lower bound of its ancestor (partial problem No. 2) is no better than the current upper bound.

**Program Realizations and Applications**

The algorithms outlined in the subsections on the branching algorithm and bounding are considered as a framework for developing efficient algorithms for solving complex scheduling problems. Although this framework is itself useful in solving scheduling problems, it is considered as a base in developing an efficient solver for complex scheduling problems by embedding additional acceleration tools. These tools can be either general-purpose, that is, valid for any problem, or specific to a class of problems. The former type of tool exploits the properties of all scheduling problems. For example, a partial problem of the B&B procedure can be pruned if all leaves that can be derived from it are cyclic. Another type of acceleration tools exploits the properties of a special class of problems. While, the problem type of independent tools is to be realized in any scheduling program based on the framework, the specific tools are to be developed for extremely complex problems whenever the general-purpose algorithm is incapable of generating the optimal solution in a realistic time period. Both types of tools may guarantee the optimality of the solution. The present article examines the program realizations of the pure framework and its acceleration with a general type of tool.

![Figure 19. S-graphs of the partial schedules generated according to the search tree given in Figure 18; longest path algorithm determines the lower bound.](image-url)
order to illustrate this framework, we solve Examples 3 and 4, downloaded from http://www.dcs.vein.hu/demo/sch/.

The recipes of the products are given in Table 1. Four equipment units, E1, E2, E3, and E4, are available to generate four products, A, B, C, and D. The recipes of the products are given in Table 1.

The proposed basic algorithm and the solution of the MILP model presented in Sanmartí et al. (1996) have been compared by solving this example in generating a different number of batches of the products. The base problem considers the production of one batch of each product, that is, four batches altogether. Then, in order to generate up to eight batches, the batches of each product were repeated once more. The results are shown in Table 2 where the MILP model has been solved with a generic solver (GAMS/OSL). For six, seven, and eight batches, the MILP did not arrive at the optimal solution in reasonable time (in several hours), or the solver crashed.

Example 4. This example was introduced by Voudouris and Grossmann (1994). Five equipment units (stages), E1, E2, E3, E4, and E5, are available to generate four products, A, B, C, and D. The recipes and the number of batches of the products are given in Tables 5 and 6. The changeover time is 60 min for equipment units E1, E2, E3, and E4, and 90 min for E5, E6, E7, E8, E9, E10, E11, E12, E13, E14, and E15; all other changeover times are zero.

The number of batches to be produced is given in Table 7 for each product; the sum of the number of batches is 33. The problem has been solved by the accelerated algorithm that required 25.27-s CPU time on a PC-Pentium 533 MHz. The resultant makespan is 7,740 min; Figure 20 shows the corresponding schedule-graph.

### Realization of the basic algorithm

The framework has been developed in C++ (it can be downloaded from http://www.dcs.vein.hu/demo/sch/). In order to illustrate this framework, we solve Examples 3 and 4.

Example 3. Four equipment units, E1, E2, E3, and E4, are available to generate four products, A, B, C, and D. The recipes of the products are given in Table 1.

The proposed basic algorithm and the solution of the MILP model presented in Sanmartí et al. (1996) have been compared by solving this example in generating a different number of batches of the products. The base problem considers the production of one batch of each product, that is, four batches altogether. Then, in order to generate up to eight batches, the batches of each product were repeated once more. The results are shown in Table 2 where the MILP model has been solved with a generic solver (GAMS/OSL). For six, seven, and eight batches, the MILP did not arrive at the optimal solution in reasonable time (in several hours), or the solver crashed.

Example 4. This example was introduced by Voudouris and Grossmann (1994). Five equipment units (stages), E1, E2, E3, E4, and E5, are available to generate four products, A, B, C, and D. The recipes and the number of batches of the products are given in Tables 5 and 6. The changeover time is 60 min for equipment units E1, E2, E3, and E4, and 90 min for E5, E6, E7, E8, E9, E10, E11, E12, E13, E14, and E15; all other changeover times are zero.

The number of batches to be produced is given in Table 7 for each product; the sum of the number of batches is 33. The problem has been solved by the accelerated algorithm that required 25.27-s CPU time on a PC-Pentium 533 MHz. The resultant makespan is 7,740 min; Figure 20 shows the corresponding schedule-graph.

### Acceleration of the basic algorithm

Even though the basic algorithm may solve scheduling problems up to a certain size, it may not be sufficiently effective for practical applications. The basic algorithm, however, can easily be accelerated by built-in combinatorial algorithms. A simple acceleration tool, the so-called loop prediction, is illustrated here. This tool tests whether an acyclic leaf can be derived from a partial problem. If the test is negative, the partial problem is to be pruned, since no feasible solution can be generated from this partial problem. This acceleration tool is realized by the loop search algorithm given in Appendix B. The program that has been developed in C++ can be downloaded from http://www.dcs.vein.hu/demo/sch/. A complex case study illustrates the behavior of the accelerated algorithm.

### Case Study

Nineteen equipment units, E1 through E19, are available to generate ten products, A, B, C, . . . , J. The recipes of the products are given in Tables 5 and 6. The changeover time is 60 min for each product, 45E3 3 51.9 62 297.5 92

### Table 4. Number of Batches for Example 4

<table>
<thead>
<tr>
<th>Product</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of batches</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 5. Recipe for Products A, B, C, D, and E

<table>
<thead>
<tr>
<th>Task</th>
<th>Eq.</th>
<th>Task</th>
<th>Eq.</th>
<th>Task</th>
<th>Eq.</th>
<th>Task</th>
<th>Eq.</th>
<th>Task</th>
<th>Eq.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>E1</td>
<td>6</td>
<td>E2</td>
<td>9</td>
<td>E4</td>
<td>8</td>
<td>E2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>E3</td>
<td>9</td>
<td>E3</td>
<td>15</td>
<td>E1</td>
<td>14</td>
<td>E3</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>E4</td>
<td>7</td>
<td>E4</td>
<td>17</td>
<td>E2</td>
<td>16</td>
<td>E1</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2. CPU Time Usage (s)

<table>
<thead>
<tr>
<th>No. of Batches</th>
<th>Proposed Basic Algorithm*</th>
<th>MILP* Santamari et al. (1996)</th>
<th>Makespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.13</td>
<td>3.86</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>1.01</td>
<td>51.9</td>
<td>62</td>
</tr>
<tr>
<td>6</td>
<td>4.48</td>
<td>—</td>
<td>73</td>
</tr>
<tr>
<td>7</td>
<td>62.24</td>
<td>—</td>
<td>87</td>
</tr>
<tr>
<td>8</td>
<td>297.5</td>
<td>—</td>
<td>92</td>
</tr>
</tbody>
</table>

* Solved by AMD K-7 Athlon 1 GHz.
### Table 6. Recipes for Products F, G, H, I, and J

<table>
<thead>
<tr>
<th>Task</th>
<th>Product F</th>
<th>Time Eq. (min)</th>
<th>Product G</th>
<th>Time Eq. (min)</th>
<th>Product H</th>
<th>Time Eq. (min)</th>
<th>Product I</th>
<th>Time Eq. (min)</th>
<th>Product J</th>
<th>Time Eq. (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E1</td>
<td>300</td>
<td>E2</td>
<td>240</td>
<td>E4</td>
<td>240</td>
<td>E1</td>
<td>300</td>
<td>E4</td>
<td>240</td>
</tr>
<tr>
<td></td>
<td>E3</td>
<td>120</td>
<td>E3</td>
<td>120</td>
<td>E3</td>
<td>120</td>
<td>E3</td>
<td>120</td>
<td>E3</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>E5</td>
<td>60</td>
<td>E5</td>
<td>60</td>
<td>E5</td>
<td>60</td>
<td>E10</td>
<td>120</td>
<td>E7</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>E7</td>
<td>90</td>
<td>E10</td>
<td>120</td>
<td>E7</td>
<td>90</td>
<td>E11</td>
<td>90</td>
<td>E9</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>E9</td>
<td>90</td>
<td>E11</td>
<td>90</td>
<td>E9</td>
<td>90</td>
<td>E12</td>
<td>60</td>
<td>E13</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>E11</td>
<td>90</td>
<td>E13</td>
<td>90</td>
<td>E13</td>
<td>90</td>
<td>E14</td>
<td>120</td>
<td>E15</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>E13</td>
<td>90</td>
<td>E14</td>
<td>120</td>
<td>E14</td>
<td>120</td>
<td>E16</td>
<td>720</td>
<td>E15</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>E16</td>
<td>720</td>
<td>E16</td>
<td>720</td>
<td>E19</td>
<td>840</td>
<td>E19</td>
<td>840</td>
<td>E17</td>
<td>540</td>
</tr>
<tr>
<td></td>
<td>E17</td>
<td>540</td>
<td>E19</td>
<td>840</td>
<td>E19</td>
<td>840</td>
<td>E17</td>
<td>540</td>
<td>E18</td>
<td>720</td>
</tr>
<tr>
<td></td>
<td>E18</td>
<td>720</td>
<td>E19</td>
<td>840</td>
<td>E17</td>
<td>540</td>
<td>E18</td>
<td>720</td>
<td>E19</td>
<td>840</td>
</tr>
</tbody>
</table>

### Table 7. Number of Batches of the Products

<table>
<thead>
<tr>
<th>Product</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of batches</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Concluding Remarks

A new framework has been proposed for job-shop scheduling based on an S-graph representation. Instead of solving an MILP model of the scheduling problem, the procedure given in the article solves the problem by specific algorithms, including a sequence of efficient graph algorithms (e.g., longest path and cycle search algorithms) that exploit the unique feature of the scheduling problem. The solution of the complex examples illustrates the efficacy of the proposed framework.

Figure 20. Schedule-graph of the optimal solution of the case study.
This framework provides the basis for further developments in solving special classes of scheduling problems, for instance, scheduling related to flexible recipes, complex recipes, and finite intermediate storage policy, which are the scope of our current work.

Acknowledgment
This research has been supported in part by the Comisión Interministerial de Ciencia i Tecnologia (CICYT) under project QUI99-1091 and by the Hungarian National Science Foundation OTKA T-029-309.

Literature Cited


Figure A1. Part of a recipe-graph: complex procedure constraints are given on the inputs of task 6.
Appendix A: Feed-Precedence Graph

Suppose that a task has multiple inputs in the recipe. In the general case, certain inputs must be fed in parallel to the equipment, others in certain orders, and some have no relation given between them. The relationship between the feeds establishes a partial order among them. This partial order can be represented by an acyclic directed graph embedded in the task network. For example, suppose that input 1 must precede input 4 and 5, and input 2 must precede input 4 in Figure A1a. Moreover, input 2 must be available continuously with input 3 in the task sequence. The additional nodes and arcs given in Figure A1b can express this order of precedence. These additional nodes are called feed-precedence nodes.

Note that for the so-called fed-batch operation, when a task has an input that must be fed continuously during the procedure of the task, the recipe-arc goes directly to the task node, bypassing the feed-precedence graph of the task if applicable (see Figure A2). The scheduling algorithm can simply take into account the feed-precedence graphs and the fed-batch operations.

Appendix B: Combinatorial Algorithms

The cycle-detection algorithm (Figure B1) is considered to be a feasible test of a partial problem of the proposed B&B algorithm, and the longest-path algorithm (Figure B2) is used in generating a lower bound of the makespan of a partial problem. More details of these algorithms can be found, for example, in Cormen et al. (1997).

Appendix C: LP Model for Determining a Lower Bound of a Partial Problem

Suppose that length $L_j$ of the longest path to node $j$ in the S-graph of a partial problem has been determined for all $j \in N$. Let $\overline{M}$ denote the set of nodes given by formula $\overline{M} = \{j: j \in N \setminus M, \forall j \in N_k \text{ for all } k \in \{1, 2, ..., m\}\} (i = 1, 2, ..., n)$. Let $c_i(i = 1, 2, ..., n)$ be defined as

$$c_i = \max \left( \max_{j \in M_i} \left( L_j + t_{ij} \right), \min_{j \in N_i/M_i} \left( L_j \right) \right) + \sum_{j \in M_i} t_{ij},$$

where $t_{ij}$ is the processing time of task $j$ by equipment unit $i$.

The solution of LP problem

$$\min X$$

s.t.

$$c_i + \sum_{j=1}^{m_i} x_{ij} \leq X \quad (i = 1, 2, ..., n)$$

$$\sum_{i \in S_j} x_{ij} \geq 1 \quad (\{j: j \in \{1, 2, ..., |N_i|\} \land |S_j| > 1\})$$

$$x_{ij} \geq 0 \quad (i = 1, 2, ..., n, j = 1, 2, ..., m_i)$$
Table C1. Recipes of Example 5

<table>
<thead>
<tr>
<th>Task</th>
<th>Product A</th>
<th>Product B</th>
<th>Product C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eq. (h)</td>
<td>Eq. (h)</td>
<td>Eq. (h)</td>
</tr>
<tr>
<td>1</td>
<td>E1 8</td>
<td>E1 9</td>
<td>E1 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E2 11</td>
<td>E2 7</td>
</tr>
<tr>
<td>2</td>
<td>E2 15</td>
<td>E3 5</td>
<td>E3 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure C1. Recipe-graph of Example 5.

The LP model will be given for the root of the enumeration tree of the B&B algorithm, that is, for the recipe-graph. Since \( M_1 = \{1\}, M_2 = \emptyset, M_3 = \{4,6\}, c_1 = 8, c_2 = 0, \) and \( c_3 = 16, \) the LP problem to be solved is

\[
\begin{align*}
\min & \quad X \\
\text{s.t.} & \quad 8 + x_{13} + x_{15} \leq X \\
& \quad x_{22} + x_{23} + x_{25} \leq X \\
& \quad 16 + x_{32} \leq X \\
& \quad \frac{x_{13}}{9} + \frac{x_{23}}{11} \geq 1 \\
& \quad \frac{x_{15}}{7} + \frac{x_{25}}{7} \geq 1 \\
& \quad \frac{x_{22}}{15} + \frac{x_{32}}{5} \geq 1 \\
& \quad x_{13}, x_{15}, x_{22}, x_{23}, x_{25}, x_{32} \geq 0.
\end{align*}
\]

The resulting lower bound is \( X = 17.4; \) it is a sharper bound than that given by the longest-path algorithm, or, 14.

Case Study Revisited. For the root of the enumeration tree, the lower bound given by the longest-path algorithm is 387 min, while it is 1,261 min when using the LP model.

Manuscript received May 17, 2001, and revision received Apr. 10, 2002.